IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 29, 2021, accepted August 13, 2021, date of publication August 18, 2021, date of current version August 27, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3105956

Explaining Individual and Collective Programming Students' Behavior by Interpreting a Black-Box Predictive Model

FILIPE DWAN PEREIRA[1], SAMUEL C. FONSECA[2], ELAINE H. T. OLIVEIRA[2], ALEXANDRA I. CRISTEA[3], (Senior Member, IEEE), HENRIK BELLHÄUSER[4], LUIZ RODRIGUES[5], DAVID B. F. OLIVEIRA[2], SEIJI ISOTANI[5], (Senior Member, IEEE), AND LEANDRO S. G. CARVALHO[2]

[1]Department of Computer Science, Federal University of Roraima, Boa Vista 69310-000, Brazil
[2]Institute of Computing, Federal University of Amazonas, Manaus 69067-005, Brazil
[3]Department of Computer Science, Durham University, Durham DH1 3LE, U.K.
[4]Department of Psychology, Johannes Gutenberg-University Mainz, 55122 Mainz, Germany
[5]Institute of Mathematics and Computer Science, University of São Paulo, São Carlos 13566-590, Brazil

Corresponding author: Filipe Dwan Pereira (filipedwan@gmail.com)

This work was supported in part by the Samsung-Universidade Federal do Amazonas (UFAM) Project for Education and Research (SUPER), according to Article 48 of Decree no. 6.008/2006 [Superintendência da Zona Franca de Manaus (SUFRAMA)], in part by Samsung Electronics of Amazonia Ltd., under the terms of Federal Law no. 8.387/1991, under Agreement 001/2020 and Agreement 003/2019, signed with the Federal University of Amazonas and Fundação de Apoio ao Ensino, Pesquisa, Extensão e Interiorização do IFAM (FAEPI), Brazil, in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brazil (CAPES) under Finance Code 001, and in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico—Brazil (CNPq) through the support to Elaine Oliveira under Process 308513/2020-7.

This work involved human subjects or animals in its research. The authors confirm that all human/animal subject research procedures and protocols are exempt from review board approval.

ABSTRACT Predicting student performance as early as possible and analysing to which extent initial student behaviour could lead to failure or success is critical in introductory programming (CS1) courses, for allowing prompt intervention in a move towards alleviating their high failure rate. However, in CS1 performance prediction, there is a serious lack of studies that interpret the predictive model's decisions. In this sense, we designed a long-term study using very fine-grained log-data of 2056 students, collected from the first two weeks of CS1 courses. We extract features that measure how students deal with deadlines, how they fix errors, how much time they spend programming, and so forth. Subsequently, we construct a predictive model that achieved cutting-edge results with area under the curve (AUC) of.89, and an average accuracy of 81.3%. To allow an effective intervention and to facilitate human-AI collaboration towards prescriptive analytics, we, for the first time, to the best of our knowledge, go a step further than the prediction itself and leverage this field by proposing an approach to explaining our predictive model decisions individually and collectively using a game-theory based framework (SHAP), (Lundberg *et al.*, 2020) that allows interpreting our black-box non-linear model linearly. In other words, we explain the feature effects, clearly by visualising and analysing individual predictions, the overall importance of features, and identification of typical prediction paths. This method can be further applied to other emerging competitive models, as the CS1 prediction field progresses, ensuring transparency of the process for key stakeholders: administrators, teachers, and students.

INDEX TERMS Explainable artificial intelligence, online judges, learning analytics, CS1, computing in education, early prediction, shapley values.

I. INTRODUCTION

Introductory programming courses (CS1) are known to have a high dropout and non-pass rate [28], [53], [58], [70]. As an attempt to alleviate that, multiple recent studies [11], [12], [16], [23], [46], [46], [47], [52], [53], [59] proposed methods to predict CS1 students' performance early on. Knowing about student performance in advance can be useful for many reasons, for example, instructors can apply specific actions to help learners who are

The associate editor coordinating the review of this manuscript and approving it for publication was Kok-Lim Alvin Yau.

struggling, as well as provide more challenging activities to high-achievers [11], [58], [67].

Previously, most methods to predict CS1 students' performance were based on a static analysis of the students' data, such as their high school grades, age, gender [1]. However, students' behaviour is dynamic and, hence, can change over time, supporting the need for data-driven analysis [11], [49], [53], [67]. Along these lines, the use of Machine Learning (ML) over data collected from e-learning systems leveraged approaches and methods to tackle the performance prediction problem [11]. Such studies tended to depict the CS1 students' behaviours based on their interaction with the e-learning systems used to support their classes [11], [12], [21], [23], [29], [32], [49]. However, the literature still lacked a reliable method to predict CS1 students' performance [58]. We thus proposed a potential solution to this gap [45], [47], [48] by composing a set of data-driven features (collected from literature and extended with self-devised ones), which we showed to have a high predictive power to infer the students' performance early on (from data from the first two weeks of the CS1 course).

A limitation of previous studies is that they did not perform any interpretation of the black-box predictive models. Thus, this paper advances the state of the art by addressing the challenge of extracting an explainable, transparent model for AI in Education for CS1 [7], by demonstrating how to interpret the predictive model's decision, in order to better support students and instructors (and other stakeholders). Such challenge is important, because the educational literature [11], [50], [53], [58] notes the lack studies on early learner behaviours that can be effective or ineffective. Effective programming behaviours are those that potentially increase the students' chances of passing, whereas ineffective behaviours decrease the students' chances of success in the course [58]. Hence, beyond the prediction, it is crucial to explain what leads the predictive model to make the decisions (e.g., why a given student *s* is classified as 'passed'), which would allow a better understanding of what early programming behaviours are to be encouraged and triggered.

Thus, in this work, our main focus is on understanding which students' early programming behaviours are related to the learner's success or failure. Moreover, we aim at analysing students' behaviours generally, to give stakeholders a high-level of early programming behaviours (*'bird's eye view'*), as well as individually, to provide an analysis of students' specificities (*'fish-eye view'*), allowing self-regulation and higher self-knowledge for the learner. To achieve this goal, we constructed a non-linear predictive model using the features of our previous works [47], [48] and we applied a game-theory based framework (SHAP method) [35], [36] that allows interpreting our black-box non-linear model linearly. The features depict useful information from fine-grained log-data collected from a home-made online judge system [49] used in our CS1 classes.

Briefly, the main contributions of this work are:

- A novel ML pipeline that focuses not only on prediction performance but also on the interpretation of the model's decision, towards Explainable AI [39], [41] and Machine Behaviour [54] study;
- Detecting, for the first time, early effective and ineffective programming behaviours at single-student granularity level;
- A new clustering approach to identify and analyse typical prediction paths for understanding of collective effective and ineffective behaviours;
- Identification and analysis of feature importance using, for the first time, instance-level explanation as building blocks.

II. CONCEPTS AND BACKGROUND

A. EXPLAINABLE MACHINE LEARNING

Nowadays, ML is mainstream, with great potential to improve education. However, predictive models often do not explain their decisions, which might be a barrier to adoption [39]. There are some simple ML methods, such as decision trees, linear regressions, decision rules, which are easily explainable [39], [41]. However, they often lack predictive power, possibly because higher accuracy for complex datasets is commonly achieved by non-linear black-box models [36], such as deep learning and ensembles [14], [24]. Consequently, a trade-off appears between performance and interpretability.

In this sense, the literature has been proposing new methods for explaining complex ML models at breakneck speed and it is often unclear how these methods are related and which one to choose [36], [39], [55]. As a response to this, [36] proposed a unified framework for interpreting predictions, SHAP (SHapley Additive exPlanations). This state-of-the-art method unifies in a single framework prestigious additive feature attribution methods such as LIME [57], DeepLIFT [64], classic Shapley value estimation [34], layer-wise relevance propagation [3] and others. Still, [9] note that, no matter the SHAP implementation used, Shapley values are challenging to interpret. Thus, as one of the contributions of our paper, we, for the first time, to the best of our knowledge, are interpreting a black-box model to better understand effective and ineffective programming student behaviours. This allows trust in early performance predicting, through transparency, as recommendations based on machines that may impact on human life need to be tractable and explicit.

SHAP is a method with foundations in game theory [63], where the features divide rewards in a way which reflects each of their contributions to the model's prediction [36], [39]. The SHAP interpretation method calculates the Shapley values for each feature at instance-level.

In practice, using SHAP we can compute the magnitude of positive or negative effects for each feature on

individual predictions. To do so, the method tests how the prediction changes when feature j is withheld from the model [36]. In other words, SHAP calculates the feature importance of a feature $j \in F$, for a given local instance x, in a given predictive model f, by evaluating the marginal contribution of that feature j for all subsets $S \subseteq F$, where F is the set of all features. Thus, the marginal contribution of j (Shapley value) is calculated by the weighted average of $f_{S\cup\{j\}}(x_{S\cup\{j\}}) - f_S(x_S)$ for all subsets $S \subseteq F$, where x_S depicts the values of the input features in the subset S. Formally, the *contribution* of a given feature j is measured by the following equation, that is the weighted average for all possible differences, computed as the combination function:

$$\phi_j = \sum_{S\subseteq F\setminus\{j\}} \frac{|S|!(|F|-|S|-1)!}{|F|!}[f_{S\cup\{j\}}(x_{S\cup\{j\}}) - f_S(x_S)]$$

where ϕ_j is the marginal contribution of feature j on the model output $f_{S\cup\{j\}}(x_{S\cup\{j\}})$. To calculate the feature contributions in a fair way, SHAP keeps fairness properties called additivity, missingness, and consistency [36], [39], [63].

Additivity means that the sum of the feature contributions together should match the output of f for the simplified input x' (which corresponds to the original input x). More formally, SHAP keeps the additivity property as:

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j . x'_j \tag{1}$$

where g is the explanation model, $x' \in \{0, 1\}^M$ is the simplified feature vector, M is the maximum simplified features vector size, and $\phi_j \in \mathbb{R}$ is the feature contribution, for a feature j, of Shapley values [36], [39]. Here, ϕ_0 represents the expected value with no prior information about the features (similar to the intercept in a regression model). In practice, ϕ_0 is the average of predictions in the training set.

The second fairness property is *missingness*. This is a trivial property, defined as:

$$x'_j = 0 \implies \phi_j = 0$$

This trivial property requires features missing in the original input to have no impact [36].

Finally, *consistency* means that if one feature contributes more to the model output, it cannot get a lower Shapley value. It is worth noting the feature contribution calculated by SHAP is the only possible explanation model that satisfies these 3 fairness properties (see the theorem proof in [36]).

In more recent work, [35] show that combining many local explanations allows capturing global patterns from the representation of the predictive model whilst retaining local faithfulness to the original model, which can be used for detailed and accurate representations of model behaviour. Figure 1 illustrates the workflow of the SHAP method, which can be used to analyse local feature effects and to combine local explanations of individual predictions, in order to generate global explanations (data insights, model summarisation, collective feature effects).

B. MACHINE LEARNING MODELS

To develop our predictive model we used the popular eXtreme Gradient Boosting method (XGBoost) [13]. XGBoost is an optimised implementation of the Gradient Tree Boosting (GTB) ML algorithm, an ensemble method based on decision trees. Specifically, XGBoost utilises the boosting principle in an iterative way, wherein at each iteration the algorithm attempts to correct the errors of the previous iteration, by optimising specific loss functions as well as applying several regularisation techniques. We opted for XGBoost, as this model has been shown to compete and even outperform standard deep learning models on tabular-style databases, such as ours, in which the attributes are meaningful and they lack strong multiscale temporal or spatial structure [13], [35]. However, it is important to experiment more ML techniques (No-Free-Lunch Theorem - ML [72]).

In previous works [45], [47], [48], we have composed a set of data-driven features that, in conjunction, have a high predictive power to infer the students' performance, even when using early data, from the first two weeks of a course. Our previous ML model [48] achieved an average accuracy of 78.2% with this early data, outperforming cutting edge results for this task [1], [12], [18], [20], [30], [33], [53], [68], [71]. Reference [48] used a state-of-the-art genetic algorithm to create an optimised shallow ML pipeline to predict student performance. Their results pointed to tree-based ensembles being more suitable for our data. As an extension, in [45] we surpassed [48], obtaining an average accuracy of 82.2%, by using a deep learning architecture. Between the model presented in this current paper using XGBoost (with average accuracy of 81.3%) and our previous best result using deep learning, we did not find statistical significance (p-value>0.05). Thus, here we can state that there are no significant performance drawbacks or advantages in our choice of XGBoost instead of a state-of-the-art Deep Learning model.

Notice that tree-based ensembles and deep neural network are non-linear techniques that construct complex models, that is, typical black-box models. As our goal is mainly interpretation, we need to 'open' such a black-box to explain the model's decision. To do so, as mentioned in the previous subsection, we used a state-of-the-art unified approach to interpret model predictions, SHAP method [36]. There are several implementations of SHAP, such as TreeSHAP, which is designed for tree-based models, and KernelSHAP, which is a model-agnostic designed for a variety of ML pipelines, such as deep neural networks. Nonetheless, [9] explain there are several caveats of KernelSHAP such as: i) KernelSHAP requires access to the entire dataset to calculate the Shapley values; ii) KernelSHAP is procedurally slower when calculating Shapley values of large datasets; iii) KernelSHAP ignores feature dependency; iv) using KernelSHAP, the Shapley values are not exactly computed, instead they are only estimated. Indeed, KernelSHAP performs a sampling of features when evaluating the possible subsets $S \subseteq F$. The TreeSHAP implementation solves all of these issues, by calculating the

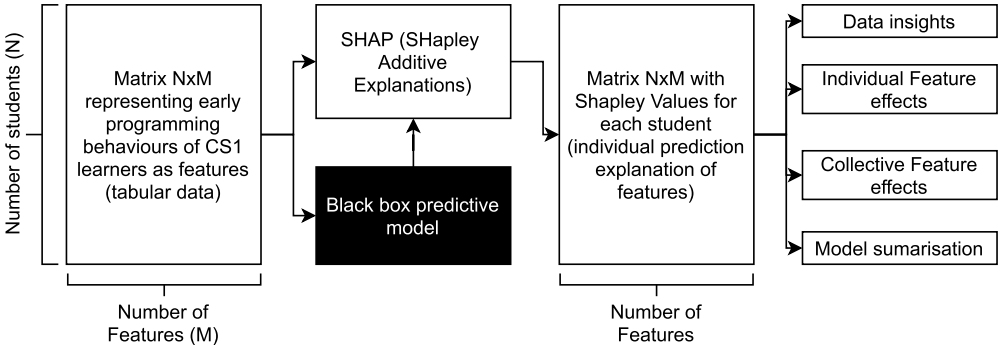


FIGURE 1. Workflow of how we used the SHAP method [35] for computing local predictions to create individual and collective explanations of the predictions from our tree-based black-box model.

exact Shapley values in polynomial time (see [35]). Thus, as a final justification for our choice for XGBoost instead of deep learning, we used this tree-based model with the TreeSHAP implementation, because it gives us more interpretative power of the models' decision, with no drawbacks with regards to the predictive model performance.

Additionally, please note that to calculate the Shapley values we need to run the predictive model many times with missing features [35], [36]. Thus, there is a need to supply a background dataset [35]. In our case, we use the training set as user-supplied background dataset, by relying only on the path coverage information stored in the tree-models, as recommended by the authors of the method [35].

III. RELATED WORK

Typically, educational data-driven researches identify patterns of behaviour based on data collected from the students learning process [4]. In general, there are many works [11], [12], [16], [23], [45], [46], [46], [47], [52], [53], [59] in this field that use ML to construct predictive models. However, there is a lack of studies that use such analyses to improve instruction and pedagogy. In other words, there is a need for a learning analytics infrastructure that provides information to support teachers and students. Recently, Carter *et al.* [11] stated that there were relevant open questions concerning what learning data should be collected within an e-learning environment for programming courses, in order to provide a foundation for improving student learning and how the learning data should be analysed to provide useful information on student learning. In this section, we will both explore how relevant works conducted studies in this direction and what we bring in terms of novelty.

First, a systematic review [28] on learning analytics using data from e-learning environments for programming classes revealed that, despite the growth of works in this field, many of them use post-hoc analysis (e.g. analysing features extracted from the environment after the class has ended) with no long-term data (as ours). The review also highlighted the necessity of reproducing and evaluating the methods and results of previously published research in other educational contexts. In response to that, we next discuss relevant works that proposed features related to programming students performance that we employed in our ML pipeline.

In terms of analysis of data collected from Integrated Development Environments (IDEs) or learning environments targeted for programming students, Jadud [30] first observed a cycle of edition, compilation and execution from a novice programming environment called BlueJ. To alleviate this, the author proposed a metric called *ErrorQuotient*, allocating higher penalty when students repeat the same compilation error. Watson *et al.* [71] propose an extension of the ErrorQuotient, called *WatwinScore*, which considers the time spent by students on the problem, for each pair of compilation errors. Estey and Coady [20] analysed other data, such as frequency of the use of hints, submissions, and compilations in an online judge. Additionally, Ahadi *et al.* [2] and Castro-Wunsch *et al.* [12] studied learners' attempts and correctness, and Leinonen *et al.* [33] analysed behaviours of typing patterns and keystroke latency. Edwards *et al.* [19] tracked the submissions of learners to evaluate the amount of change between code submissions, and procrastination behaviours. All these studies analysed the relationship between programming students behaviours and their performance, mainly by using ML techniques to construct predictive models. However, such works did not explain why these code metrics were helpful to predict the students' performance and how they can be useful to improve their learning and instructions.

Importantly, when predicting students performance, it is vital to do so as soon as possible, in order to allow an early intervention [23], [29], [32], [53], [59]. With this in mind, many features proposed by the aforementioned works [2], [2], [12], [19], [20], [30], [33], [71] reported a generally low predictive power at the beginning of the course (e.g., based on data gathered from the first two weeks). However, we have lately improved this by reporting a high predictive power in our most recent previous work [45], [48]. To achieve this, we collected not only all programming behaviour indicators from past research that could be applied in an educational context [2], [2], [12], [19], [20], [30], [33], [71], but also

proposed additional self-devised features as input to machine learning models, to predict, based on early data, whether students would pass or fail. In the current paper, we extend these works [45], [48], by using our previously proposed features and database, not only with early prediction purpose, but also with the goal of interpreting the model's decision, to create the foundation for improving student learning and to provide useful information for stakeholders related to CS1 classes.

In brief, what is already known about this topic is that data collected from programming environments (e.g., IDEs or online judges) can be used to predict students performance; however, there is a need for interpretation of these programming behaviours, firstly, to justify potential recommendations, to allow tracing of the decision process, and to underpin responsible decisions. Ultimately, we aim to support students' learning process and teachers' instruction and pedagogy. In this sense, we go further than the early prediction task, by analysing which general behaviours of programming students are desirable and which need to be improved. Moreover, we explore and interpret individual student programming behaviour, which could allow the learner to reflect on what they are doing and how likely it would be for this to lead to success or failure.

IV. RESEARCH DESIGN

A. TEACHING SETTINGS

The CS1 course is compulsory to 16 STEM degrees at the Federal University of the Amazonas (UFAM) that do not have Computing as their major. Learners originate thus from non-CS courses from five major areas: Mathematics, Physics, Engineering, Statistics and Geology. Specifically, three of the degrees belong to Mathematics, two to Physics, eight to Engineering, one to Statistics and one to Geology. In particular, CS1 is offered during the first term in 11 of these 16 courses, whilst offered in the second term for the other 5 courses. In such a situation (non-CS students learning to program), the literature [17], [22], [61] suggests that some students may be less motivated to learn, as they might fail to see the purpose that programming can have in their professional lives. Our own observations confirm this: we have collected learners' data from 2016 to 2018. Generally, we can notice a high non-pass rate (about 50%).

The CS1 instructors have applied various methods to improve this situation. For instance, since 2015 they have adopted a blended learning methodology to encourage students to learn programming 'by doing' [58]. That is, students needing to solve many problems to improve their skills. However, besides practising, it is vital that the students receive quick feedback, in order to locate their errors, understand their source and fix them. Ihantola *et al.* [28] state that the continuous evaluation during a programming course ensures that students practice more, as long as they receive feedback on the quality of their solutions. This happens because the assessment guides learning and serves as *feedback* for both the student and the instructor. Nonetheless, only increasing the number of problems for students to solve also increases the instructors' workload in correcting the learners' solutions. To illustrate, in an assignment with 10 questions for a class of 100 students, the instructor would need to evaluate 1000 pieces of code. Thus, we opted for using a way of automatically evaluating students' codes, in order to offer them instantaneous feedback. To do so, as well as to be able to both trace all student progress and improve the offer based on it, we built and used a home-made online judge called CodeBench (see Figure 2), created by one of the authors, to support the blended learning methodology. CodeBench offers similar features as other online judges (e.g. URI [8], Jutge [51], UVa [56]), exercises comprising a variety of programming skills, such as code tracking, error identification and correction, code building, code reuse, among others, as recommended by the programming research literature [26]. In CodeBench, among other facilities, students solve the problems in an embedded IDE and receive instantaneous feedback about their code solution.

CS1 classes at UFAM involved 60 classroom hours, equivalent to 30 lessons, and were organised to start with an introductory module of two lessons, allowing learners to familiarise themselves with CodeBench; followed by seven thematic modules, containing four lessons each. The first and fourth lesson of each thematic module was compulsory and run face-to-face, whilst the second and the third lesson was optional and online, requiring the use of CodeBench. The thematic content included: variables and sequential structure, conditionals, nested conditionals, while-loop structures, arrays and strings, for-loop structures, and bi-dimensional arrays. The sessions were structured so that questions became gradually harder. Each session had the following sequence of activities: one opening lecture, two practice classes, and an exam. All face-to-face classes were held in computer laboratories. The examples and exercises discussed in each session were cumulative. To illustrate, the main topic of the fourth module is while-loop structures; however, it includes concepts from all previous sessions. The final grade was calculated based on 7 partial exams, 7 assignments, and one final exam. The grades from the partial exams had increasing weights (6.1% to 18.2%) towards the final grade. The grades from all assignments had the same weight on the final grade (1.3%), as students were aware. Students solved programming problems from the assignments and the exams using Python.

B. DATA COLLECTION

For the prediction and analysis in this paper, we use data[1] from 2058 students over 6 semesters (2016-2018), as CodeBench was only introduced in 2016. During this period, these students were exposed by CodeBench to 893 different problems during assignments, and to 107 problems during exams. In total, the learners submitted

[1]A description of the data, examples of test codes and logs, and the dataset used in this study can be found in codebench.icomp.ufam.edu.br/edu_dataset/

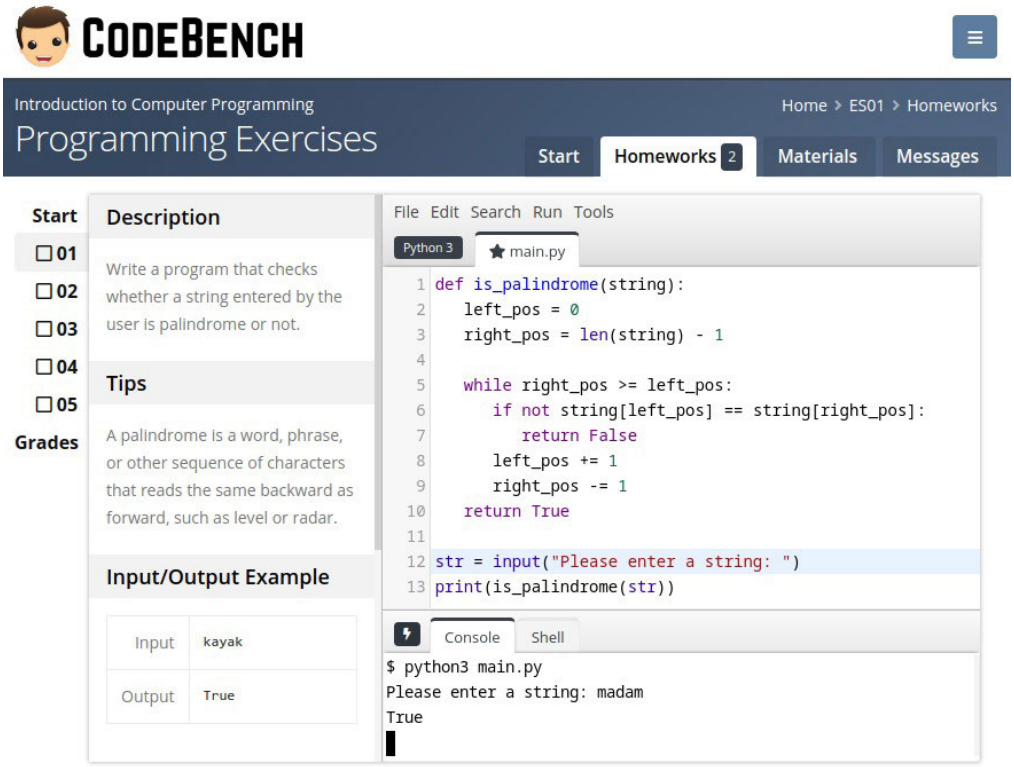


FIGURE 2. Screenshot of the student interface of CodeBench, showing a programming assignment that comprises 5 exercises (left), the description and the tips of question 01 (middle), the embedded IDE with an example of code (right), the menu (right top), and the Python console, running this program (right bottom).

```
2016-7-28 10:04:18.826#change#{"from":{"line":0,"ch":0},"to":{"line":0,"ch":1},"text":[""],"removed":["n"],"origin":"+delete"}
2016-7-28 10:04:18.826#keyHandled#"Backspace"
2016-7-28 10:04:42.332#change#{"from":{"line":0,"ch":2},"to":{"line":0,"ch":2},"text":["v"],"removed":[""],"origin":"+input"}
2016-7-28 10:04:42.492#change#{"from":{"line":0,"ch":3},"to":{"line":0,"ch":3},"text":["i"],"removed":[""],"origin":"+input"}
2016-7-28 10:04:42.677#change#{"from":{"line":0,"ch":4},"to":{"line":0,"ch":4},"text":["r"],"removed":[""],"origin":"+input"}
2016-7-28 10:04:42.828#change#{"from":{"line":0,"ch":5},"to":{"line":0,"ch":5},"text":["u"],"removed":[""],"origin":"+input"}
2016-7-28 10:04:42.988#change#{"from":{"line":0,"ch":6},"to":{"line":0,"ch":6},"text":["s"],"removed":[""],"origin":"+input"}
2016-7-28 10:04:43.635#change#{"from":{"line":0,"ch":7},"to":{"line":0,"ch":7},"text":["",""],"removed":[""],"origin":"+input"}
2016-7-28 10:04:43.635#keyHandled#"Enter"
```

FIGURE 3. Example of log data collected from the student in Figure 2, when solving a programming problem in CodeBench.

150,314 pieces of code, as solutions to our assignments/exams.

For analysis, we perform a fine-grained source code snapshot data collection annotated with events, such as tests, submissions, executions, keystroke, and so on, extracted from our online judge. To illustrate, we stored each student action of inserting or removing characters, pasting text, mouse right-clicks, etc. Additionally, each log is timestamped.

For a better understanding of the data used in this work, in Figure 3 we show sample logs collected from CodeBench, when a student was writing a Python instruction in the embedded IDE (see Figure 2).

C. REPRESENTATION OF STUDENT PROGRAMMING BEHAVIOUR

Reference [11] proposed a taxonomy to classify the way to extract useful information from user data, based on three levels. The first, *Count*, represents features that can be extracted by counting events in raw log files or source codes; *Math* represents features that need a mathematical formula to be computed; and *Algo* represents those features that need an algorithm applied. In this sense, Table 1 shows all features (programming behaviours) used in our new ML model (which were extracted from the data presented in section IV-B), as well as their description and classification using the taxonomy from [11]. All these features were validated in our previous works [45], [47], showing high predictive power for early performance prediction.

D. DATA CLEANING

Firstly, the data from registered students that did not attend the course was removed from this analysis, since they did not have any interaction with the online judge. Second, we collected data from the very beginning of the course to predict whether the student will pass or not. In the meanwhile, a student might change her/his attitude in the course.

TABLE 1. Features (programming behaviour) used in our ML models [45].

Features (Programming behaviour)	Description	Type
procrastination	Time (in days) between first code edit and programming assignment deadline [11], [19] multiplied by -1 after the z-score transformation;	Count
amountOfChange	Amount of code added/changed between submissions [11], [19];	Count
attempts	Average number of submission attempts (regardless of whether correct or not) for each problem [12];	Math
lloc	Total number of logical lines for each submitted code [42]. Imports, comments, and blank lines were not counted;	Math
systemAccess	Total number of student logins between the beginning and the end of the second week;	Count
firstExamGrade	Student grades for the first exam taken at the end of the second week of the course;	Count
events	Number of log lines on the attempt to solve problems. To illustrate, each time the student presses a button in the embedded IDE of CodeBench, this event is stored as a line in a log file (adapted from [12], [33])	Count
eventActivity	A binary self-devised feature, where 1 is assigned when the student solves a problem with less than an amount* of *logRows*. To aggregate the feature, we calculate the probability of a student having a value 1;	Algo
correctness	Number of problems solved correctly from the programming assignment realised in the first four weeks [12];	Count
correctnessCodeAct	Represents the same as correctness, but in this case, we consider 'correct' only student's solutions with more than 50 events. To illustrate, if a student submits a correct solution by copying and pasting (only 1 event), for this problem it will be assigned 0 to the feature correctnessCodeAct, however for the feature correctness will be assigned 1;	Algo
copyPaste	A self-devised feature, the proportion between pasted characters ('ctrl+V') and characters typed;	Math
syntaxError	A self-devised feature, ratio between the number of submissions with syntax error** and the number of attempts [20];	Math
ideUsage	Total time spent, in minutes, by the students solving problems in the embedded IDE (counted only when students were typing - we removed downtime);	Algo
keystrokeLatency	Keystroke average latency of the students when typing in the embedded IDE (we also removed downtime) - adapted from [33];	Algo
errorQuotient	Compute a score based on the number of code errors and repeated errors [30] (more explanation in related work - Section III);	Algo
watWinScore	An extension of errorQuotient taking into account the problem solving time [71] (more explanation in related work - Section III);	Algo
deleteAvg	Average of deleted characters for each problem;	Count
comments	Total number of comment lines on source codes;	Count
countVar	Total number of variables in the source codes [11].	Count

*One standard deviation minus the median of the numbers of *logRows* for a problem
**SyntaxError is a common and generic exception in python.

To illustrate, a learner might start the course engaged, doing the exercises and interacting in an effective way with the online judge. However, for some reasons outside of our control, the student might get disengaged during the course and end up failing. Such a change will potentially produce a false positive for our predictive model. Notice that the same reasoning can explain false negatives, where a learner might start with ineffective behaviours, but change and end up passing. Third, as we are dealing with features extracted from very fine-grained log-data, collected from students' interactions with an online system, this might cause server-side problems. For example, if a student loses internet connection while solving a problem on the IDE, then her/his logs will not be sent properly to the server. As such, our database might have some outliers.

Aiming to decrease the biases of the classifiers due to the presence of outliers, we automatically removed 100 instances from the majority class (students who passed) that form Tomek's links [69]. This approach using Tomek's links is recommended in situations like ours, requiring undersampling for balancing [5]. Our database is subtly unbalanced, having 56.7% of students who passed and 43.3% who failed, so we removed from the majority class. Thus, our data cleaning process does not only remove potential outliers, but at the same time, it reduces the unbalanced nature of the dataset, which also decreases the biases of the classifiers.

A Tomek's link is defined between two samples x_i and x_j of different classes $c1$ and $c2$, respectively, such that, for any sample y, $d(x1, x2) < d(x1, y)$ and $d(x1, x2) < d(x2, y)$, where d(.) is the Euclidean distance between the two samples. Hence, a Tomek's link is represented by two samples from different classes that are each other's nearest neighbours, which might confuse the ML model when creating the decision boundaries to separate the instances of classes [5].

To further deal with other balance aspects of the dataset, we next divided the instances into homogeneous subgroups called stratum (stratified sampling), so that the right number of instances is sampled from each stratum, in order to keep the same class proportion in the training and validation sets [24]. To do so, we used the *StratifiedKFold* from *scikit-learn* [44], using a total of 10 folds.

Finally, in ML, it is important that, once the features are selected, they are all mapped onto a similar scale, for a fair way of processing them together [24]. Here, we standardised

the features using the popular z-score (z_i), i.e., assigning zero to the mean and replacing values x_i with the number of standard deviations $\sigma(x_i)$ from this mean. This transformation was used, as it allows an arguably 'natural' predictive model interpretation, by relying on the mean and the standard deviation, to verify whether a programming behaviour (Table 1), represented by a feature value, might be effective or ineffective for a given student.

E. CLASSIFICATION AND VALIDATION

In this work, as said, we used XGBoost, which has parameters to control the ensemble training, such as the number of trees (*n_estimators*), as well as parameters to control the growth of trees (e.g., *max_depth*, *min_samples_leaf*, etc.). Moreover, an important parameter is the learning rate, which scales the contribution of each tree. To train our model, we used a popular regularisation technique called *shrinkage* [24], in which we set a low value to the learning rate (e.g. 0.05), and a high number of decision trees (100 estimators). Finally, we used the *early-stopping* technique with at most 100 rounds, meaning that we stopped training when the validation error stopped decreasing, to avoid overfitting.

As a baseline state-of-the-art for our model, we used the deep learning model presented in [45] and the promising classifier using genetic algorithms in [48]. The deep learning model was a feed-forward Multilayer Perceptron (MLP) with two hidden dense layers with 64 nodes each. As activation function, RELU was used and the biases and weights of the MLP were initialised randomly, following a normal distribution. Additionally, [45] used 50% of dropout to avoid overfitting and ADAM was used for the gradient descent optimisation. On the other hand, a regularised Random Forest (RF) with 100 estimators was found as one of the best models by the genetic algorithm in [48]. Notice that these recent works [45], [48] achieved state-of-the-art results for the classification task of early performance prediction of introductory programming students.

In closing, the models were evaluated using several statistical measures, to ensure a comprehensive picture of the results: recall, precision, F1-score, ROC curves, and accuracy.

V. RESULTS AND DISCUSSION

A. CHARACTERISATION OF STUDENTS' PROGRAMMING BEHAVIOURS

For a general picture of the original values from the programming behaviours used as features by the ML models, we first show distributions of these features in Figure 4, before the z-score transformation. Overall, we can observe the lack of symmetry in most of the cases, by seeing that most features have high positive skewness ($skewness > 1.0$) (*procrastination*, *amountOfChange*, *comments*, *systemAccess*, *events*, *copyPaste*, *syntaxError*, *ideUsage*, *deleteAvg*, *errorQuotient*, *watWinScore*) with long tails ($kurtosis > 1.0$), which means that they tend to be concentrated in lower values of the distribution. Indeed, only the *eventActivity* has a high negative skewness ($skewness < -1.0$), which means the values tend to be concentrated in the higher values. On the other hand, the other features (*attempts*, *lloc*, *firstExamGrade*, *correctness*, *correctnessCodeAct*, *keystrokeLatency*, *countVar*, *deleteAvg*, *finalGrade*) have low or moderate skewness. Moreover, we notice an overall high variation in the features, which indicates heterogeneity in the students' behaviours.[2]

B. PREDICTIVE MODELS COMPARISON

We constructed our predictive model using XGBoost, as described in subsection IV-E. It is intrinsic for XGBoost to automatically select the best feature as the root of the constituting tree of the predictive model (and, similarly, for sub-trees); using this algorithm performs an automatic feature importance analysis [13]. As such, we opted for not using any additional feature selection technique in our work.

To analyse the competitiveness of our XGBoost model results, we compared them to the current state-of-the-art works [45], [48]. For each predictive model, we ran the stratified cross-validation 20 times with 10 folds (as recommended by [18], [45]), varying the seed in a range from 1 to 20, in order to shuffle the database in different ways, to ensure reliable results. Hence, we report outcomes from the 200 results for each metric (20×10, one outcome for each fold). All models were trained using data from the very first two weeks of the course, for early prediction.

The results of each method are presented in Table 2, where Random Forest (RF) is the model found by the genetic algorithm in [48] and DL is the deep learning model found in [45]. Our predictive model (XGB) achieved an accuracy ranging from 81.1% to 81.6% (C.L. 95%). Indeed, our current model statistically significantly surpasses [48], even with Bonferroni correction (p-value $< 0.05/3$) in all evaluation metrics (accuracy, F1-score, precision and recall). Moreover, our XGB model surpassed the DL model presented in [45] in terms of precision, whilst it is surpassed by the DL model in terms of accuracy, and there is a draw for the other metrics (F1-score and recall). Still, this difference is not statistically significant for the F1-score (p-value $= 0.625$) and recall (p-value $= 0.05$), whilst there are statistical differences for accuracy (p-value $< 0.05/3$) and precision (p-value $< 0.05/3$). Notice that we are dealing here with a database that is slightly unbalanced and, hence, our XGBoost may have some advantage, since the XGB model achieved higher results for precision, and accuracy might be misleading, even for such subtly unbalanced databases [5], [24]. Moreover, we can also see that our XGBoost model is more stable in terms of accuracy, since the standard deviation and interquartile range of accuracy are lower than in the DL model. In addition, as explained in section II-B, we opted for XGBoost because TreeSHAP [35], which is designed for tree-based models, solved several issues of KernelSHAP [36], which is typically used for DL models.

[2]For more details about the distributions of the programming behaviours, see Appendix A.

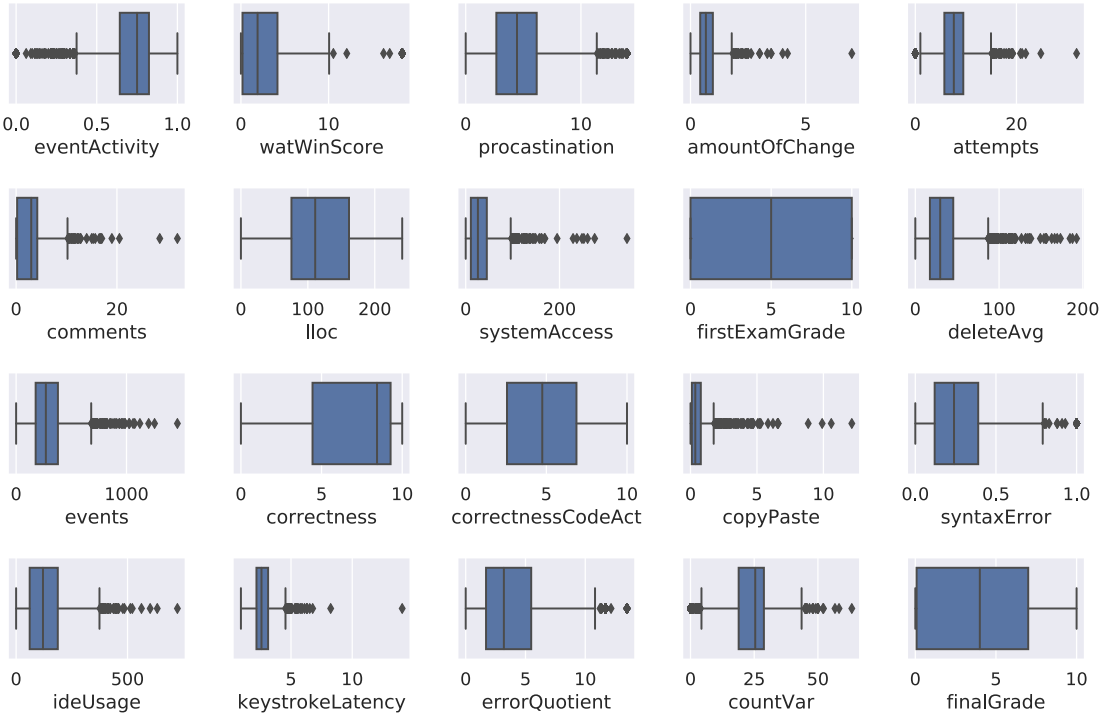


FIGURE 4. Distributions of programming behaviours (features) and distribution of our dependent variable (final grade) before discretisation.

TABLE 2. Comparison of our prediction model and our baselines.

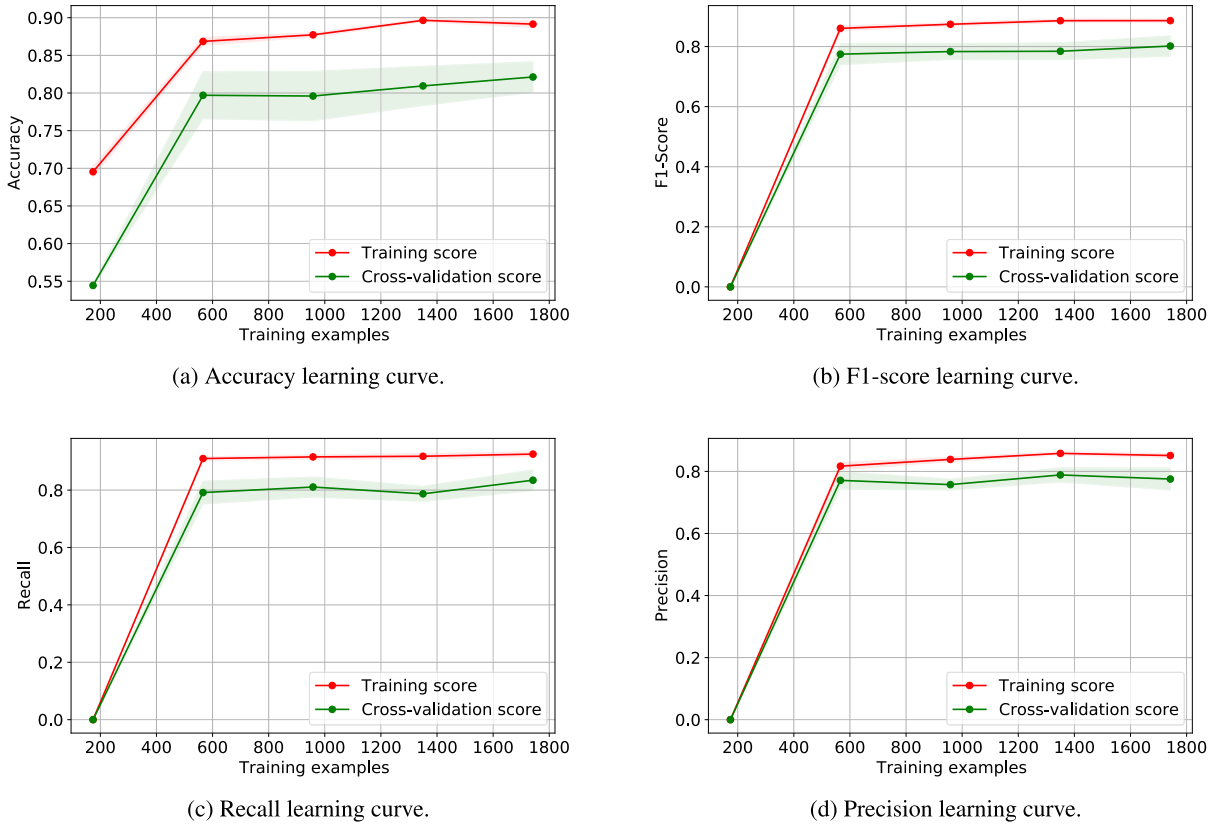
		Accuracy			Recall			F1-Score			Precision		
		XGB	DL	RF	XGB	DL	RF	XGB	DL	RF	XGB	DL	RF
Mean		0.813	**0.823**	0.797	0.850	**0.860**	0.838	**0.821**	0.818	0.792	**0.794**	0.782	0.751
95% C.I. for Mean	Lower Bound	0.811	**0.819**	0.794	0.844	**0.854**	0.833	**0.816**	0.814	0.789	**0.791**	0.777	0.747
	Upper Bound	0.816	**0.827**	0.800	0.857	**0.865**	0.843	**0.826**	0.822	0.795	**0.797**	0.787	0.756
Median		0.809	**0.827**	0.798	0.841	**0.864**	0.835	0.809	**0.821**	0.792	0.797	0.781	0.747
Std. Deviation		**0.018**	0.027	0.022	0.049	0.039	**0.034**	0.033	0.027	**0.022**	0.023	0.037	0.032
Minimum		**0.774**	0.754	0.742	**0.787**	0.739	0.766	**0.775**	0.746	0.746	0.750	0.692	0.684
Maximum		0.839	**0.895**	0.855	0.936	**0.966**	0.915	0.882	**0.888**	0.847	0.835	0.886	0.830
Interquartile Range		**0.031**	0.037	0.030	0.079	0.053	**0.047**	0.047	0.037	**0.028**	0.022	0.052	0.038

Moreover, when analysing other relevant works that also have the goal of early performance prediction in introductory programming, [33] achieved an accuracy of 65.8%, using data from 226 students in their first two weeks of the course. In an extension of that work, using more data, they achieved 72% accuracy [18]. Using data from 897 learners in their first four weeks,[3] Reference [12] achieved an accuracy of 71.81%. [53] achieved an average accuracy of 71% using early data from 692 students. Reference [68] achieved 78% of accuracy. Indeed, our result is superior to all related works that performed early performance prediction (section III). Although all these works were conducted in different educational scenarios, their performance provides us with the intuition that the problem of early prediction is complex. Nevertheless, our XGBoost model achieves high performance. Moreover, we are the first, to the best of our knowledge, to apply explainable artificial intelligence to interpret the predictions of the predictive CS1 model's decision.

[3]The performance in the first two weeks is not reported in [12].

C. RELIABILITY OF THE XGBOOST MODEL

To demonstrate the reliability of our XGBoost model, we analysed its learning curves for an increasing number of instances (students) using 10 fold cross-validation, as recommended by the literature [24]. We plotted the average cross-validation performance (analysing a comprehensive set of evaluation measures: accuracy, F1-score, recall, and precision) and the standard deviation in the shaded areas of Figure 5. We started with 180 instances and then incremented in increments of 400 instances, for the cross-validation of our XGBoost. We stopped this process at 1760 (as one more increment would exceed our total number of our instances). Notice that from 580 instances on, the predictive model



(a) Accuracy learning curve.

(b) F1-score learning curve.

(c) Recall learning curve.

(d) Precision learning curve.

FIGURE 5. Learning curve of our predictive model for different metrics (accuracy, F1-score, precision and recall) taking into consideration varied numbers of instances for training.

achieves a score close to 80% in all performance measures. As such, 580 instances is potentially the number of students needed for convergence, which endorses the possibility that our model could be used even for a lightweight database. Moreover, from a visual inspection of the plots, we can observe that our model generalises well on the validation set, as the continuous lines (red and green) are close to each other, which indicates that our model did not overfit the training set.

Furthermore, analysing the trade-off between bias and variance, we can state that our model potentially found a balance between these errors, since the variability around the training score and cross-validation score curves are almost stable from the convergence point and the curves are similar (see [24]).

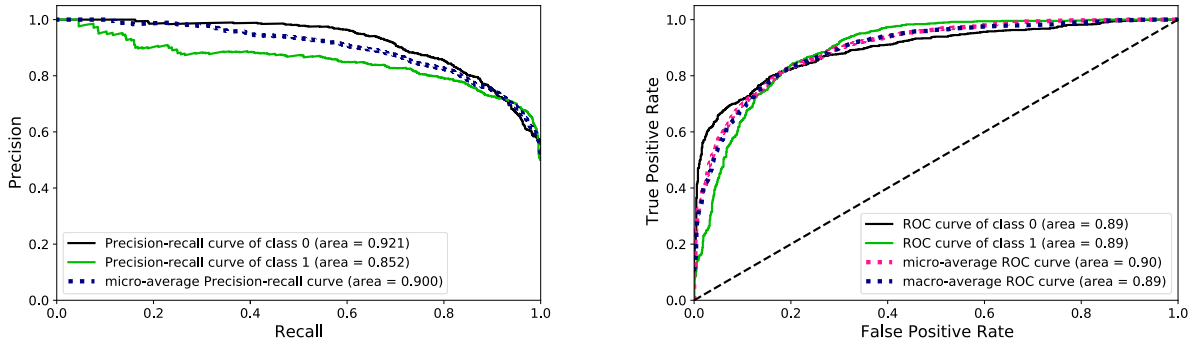
Next, Figure 6 shows the precision and recall curve (a) and ROC curve (b) of students who passed and students who failed on the validation sets. The micro-average takes into consideration the class proportion of students who passed and failed, whereas the macro-average treats each class independently. In Figure 6 (a), the recall and precision curves are plotted for different thresholds, whilst in Figure 6 (b) (ROC curves) the false positive rates and the true positive rate are also plotted for different thresholds, where positives are represented by students who passed and negative by failed students. The area under the curve achieved was 0.89 (for both classes) and the precision/recall curve obtained was 0.85 for class 1 (passed) and 0.92 for class 0 (failed). These results indicate that the binary classifier segregated students well, even when the threshold was different from the central value. This can be seen by analysing how close the continuous lines (green and black) are in both graphs from Figure 6.

D. INTERPRETATION OF THE PREDICTIVE MODEL

As previously argued, obtaining an accurate predictive model is important, but not enough, if its decisions are obscure - especially when working with a vulnerable population such as that of learners. Thus, after ensuring a competitive performance of our model, we show here how to interpret its decisions. Next, we use the SHAP method (as explained above) to explain effective and ineffective behaviours behind individual predictions, *prediction paths*, and collective behaviours.

E. INDIVIDUAL ANALYSIS

To provide a general idea of how we can evaluate which learner programming behaviours are effective and ineffective, we can inspect graphically the Shapley values of each learner, individually. Figure 7 shows decision plots with coloured lines (vertical), where a light brown line represents an individual prediction of a student that failed in CS1, whilst a dark purple line depicts a student who passed in the course. The students were chosen at random. The x-axis represents the model's output: in this case, the probability of a student



(a) Precision and recall curve.

(b) ROC curve.

FIGURE 6. **Precision and recall curve (a) and ROC curve (b) of students who passed (class 1) and students who failed (class 0).**

passing.[4] The students' coloured line cross the (top) x-axis at our model's predicted probability value. To classify the students, we used as threshold the *base value*,[5] which is approximately 0.46. Hence, if the probability of students passing is higher than this threshold, then they will be classified as passed, otherwise as failed. To illustrate, in Figure 7 (a), our model predicted that the probability of the highlighted student passing is close to 0.05 (5%) and, hence, the student is classified as failed, whilst in Figure 7 (b), the probability of the highlighted learner passing is close to 0.85 (85%), thus classified as passed.

The y-axis of the decision plot lists our features in descending order of importance. Each feature's importance is specific for the student plotted in that particular decision plot. Moreover, the straight vertical grey line marks the model's base value. Finally, from the bottom to the top of the plot, the decision plot shows cumulative Shapley values (feature effects - see section II) for each student's programming behaviour, i.e., for a given prediction we show how each student's programming behaviour (feature value) contributes to the overall prediction over the model's base value. We also show the feature values next to the coloured student prediction line, for reference. Remember that the feature values are standardised with the z-score, representing, for each feature, how far a student is from its mean value.

1) EXPLAINING INDIVIDUAL PROGRAMMING BEHAVIOURS OF A LEARNER WITH A HIGH CHANCE OF PASSING

With information as above, we can perform a deep explanation of individual predictions, i.e., we are able to uncover notable patterns of programming behaviours that can be useful for a better understanding of what might lead to success or failure. In Figure 7 (b), we can see that this student has a high probability of passing (≈85%).

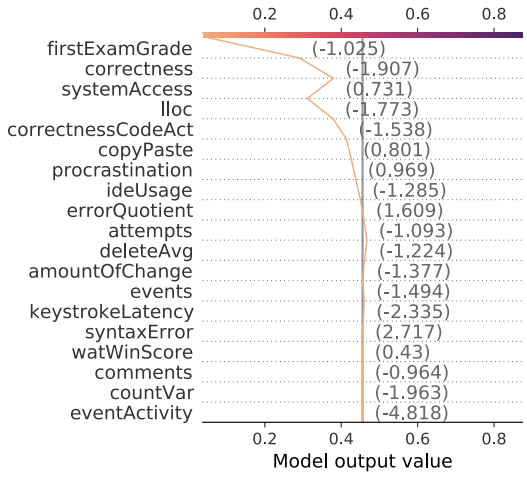
Observing the decision plot, we can notice that the features *eventActivity*, *attempts*, *watWinScore*, *countVar*, *syntaxError*, *events*, and *keystrokeLatency* had no effect for this learner. Indeed, feature values that push the prediction higher (effective behaviours) are the learner's moderately low *deleteAvg* (−0.5), low *copyPaste* (−1.1), moderately high *correctnessCodeAct* (0.72), moderately low *errorQuotient* (−0.6), average *systemAccess* (0.1), average *ideUsage* (−0.2), low *procrastination* (−1.0), moderate high *lloc* (0.81), high *correctness* (1.0) and high *firstExamGrade* (1.1). Considering the effective behaviours of this student, we notice that the student deleted parts of their code less frequently than her/his peers, which might indicate that this student is not struggling, or rewriting the code many times. This can also be seen by observing that the negative *errorQuotient* increases the student's chances of passing. Moreover, s/he makes low use of *copyPaste* and, hence, s/he is potentially writing the code from scratch. Finally, s/he achieved a high grade in the first assignment list and exam. On the other hand, the features that push the prediction lower (ineffective behaviours) are the high value of *comments* (1.0) and *amountOfChange* (1.0), which is somewhat unexpected. A high value of *amountOfChange* as ineffective might be explained by the fact that this learner has a moderately low *errorQuotient* and, thus, theoretically, would not need to make many changes between submissions. About the *comments*, it seems to be a hidden pattern that the predictive model uncovers for this learner. That is, a high number of comments in the beginning of the course is not increasing the learner's chances of passing. This may potentially be because the (Python) code required is too easy and brief at this point of the CS1 course (first two weeks), without great need of documentation. Mnemonic variable names might be enough to explain such code.

2) EXPLAINING INDIVIDUAL PROGRAMMING BEHAVIOURS OF A LEARNER WITH A HIGH CHANCE OF FAILING

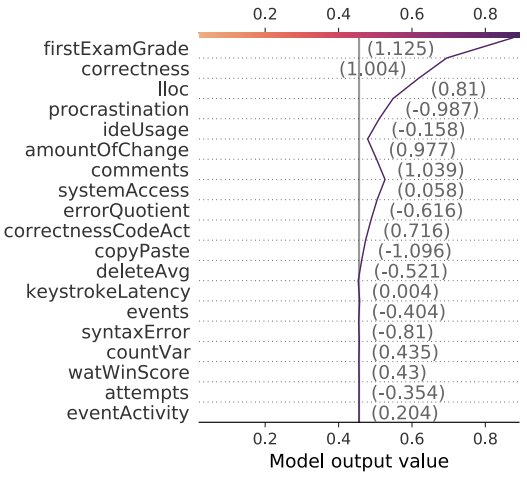
Conversely, in Figure 7 (a) we show an example of a student who has a high chance of ending up failing. Overall, this learner has ineffective behaviours, such as a high

[4] In order to calculate the probability of failing we just subtracted: $(1-p_p)$, with p_p probability of passing.

[5] The base value is the average prediction over the training set. This value represents the overall value that would be predicted if we did not know any features of the current output [36].



(a) Example student-level decision plot of student predicted to **fail**.



(b) Example student-level decision plot of student predicted to **pass**.

FIGURE 7. Decision plots to explain the potential leading factors (early programming behaviours) for passing or failing.

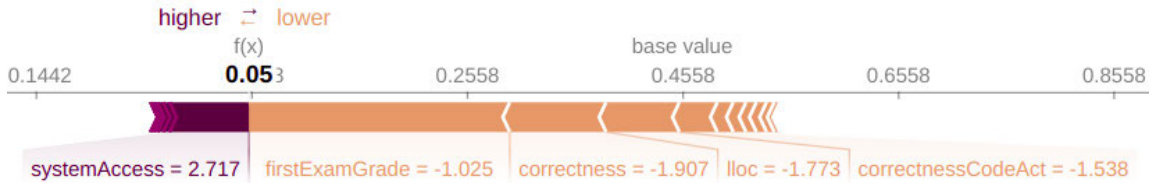


FIGURE 8. Force plot of a given student.

errorQuotient (1.6), low *ideUsage* (−1.3), high *procrastination* (1.0), and moderately high *copyPaste* (0.80). Moreover, s/he has low correctness (−1.9), *correctnessCodeAct* (−1.5), and *firstExamGrade* (−1.0). As an effective behaviour, s/he accesses the system more than the average: *systemAccess* = *0.7*. Thus, we can assume that this learner expends little effort in trying to solve the problems from the assignment. Some indicators of that are the low *ideUsage*, high *procrastination*, high *copyPaste*, and low *correctnessCodeAct*.

We can also visualise an individual explanation of the model prediction as a force plot [37], presented in Figure 8. Similarly to the decision plot, the force plot presents a prediction for a student (here, chosen at random). The $f(x)$ function is the model output (the predicted probability for that student), and the base value follows the same reasoning of the decision plot (average of model predictions). The features that push the prediction higher are shown in dark purple, whilst the ones which push the prediction lower are in light brown. To be more meaningful, the dark purple features are right arrows, whereas the light brown ones are left arrows. The arrow's size represents the effect of that feature. Given that, from a visual inspection of Figure 8, we can observe that the leading factors that are pushing the prediction lower are that s/he has a low *firstExamGrade* (−1.02), *correctness* (−1.9), *lloc* (−1.7), and *correctnessCodeAct* (−1.5). However, similar to the learner from Figure 7 (a), s/he has a high *systemAccess* (2.7) value, which is slightly increasing the learners' chance of passing.

Based on such individual explanations, we can generate automatic, customised, fine-grained suggestions to a student; or provide this detailed information to the teacher, who can use it in talking (face to face) with the student, to encourage the learner to better use her/his potential; e.g., guiding them towards being more hardworking - by solving more problems from scratch, and not too close to the deadline. As another example where there is room for improvement, some learners are copying and pasting more than 1 standard deviation above the average, which might not be a desirable behaviour for a novice student in the first two weeks of the course. Instead, it is expected that students solve problems from scratch, to practice more, as recommended by the testing effect theory [60], which explains the role of effortful processing as a contributor to the achievement.

Notice that although the force plot might seems more intuitive for interpretation, it is useful only for a few features, while the decision plot can present a large number of features effects clearly. Moreover, in a decision plot, we can visualise multi-output predictions, as we show in the next subsection, which allows detecting some prediction paths.

F. SMALL GROUP ANALYSIS (PASSED VERSUS FAILED)

After an individual analysis of student behaviours, we join 10 low-achieving students who failed, in Figure 9 (a), and 10 high-achievers students, who passed, in Figure 9 (b), to inspect patterns related to the predictions. All students were chosen at random. Such local explanations can be useful, as building-blocks for global insights. Here we notice that the failing students have a similar trajectory (prediction paths), that is, their learning lines are relatively close for many features, which shows a similarity in their programming behaviours. However, we can see some exceptions. To illustrate, we can observe a student who crossed the margin line, which suggests that this learner was performing well towards passing the CS1 course, e.g., s/he did not procrastinate too much, accessed the online judge (*systemAccess*) regularly, with a medium number of events and *eventActivity*. Nonetheless, s/he made many mistakes while submitting the code (see *errorQuotient*, *watWinScore* and *syntaxError*) and solved a lower number of problems from the assignments (lower *correctness*), and then, perhaps for some an unknown reason (extraneous variables), ended up failing.

Another observation for prediction paths of the successful students (Figure 9 (b)) is that we note two divisions in the plot: (i) the first is for students who did not struggle too much, which is illustrated by the lines which have often been above the vertical line margin; (ii) the other students have encountered higher difficulty, but they were still successful. The lines of these students are on both sides of the margin line.

From this small sample of learners, although we can observe similarity in prediction paths of the successful and unsuccessful learners, there are some nuances in the behaviours that might lead to success or failure in this cohort. In the next subsection we will evaluate these nuances in the prediction paths more holistically, taking into consideration almost the entire dataset, instead of a small sample of learners.

G. PREDICTION PATHS

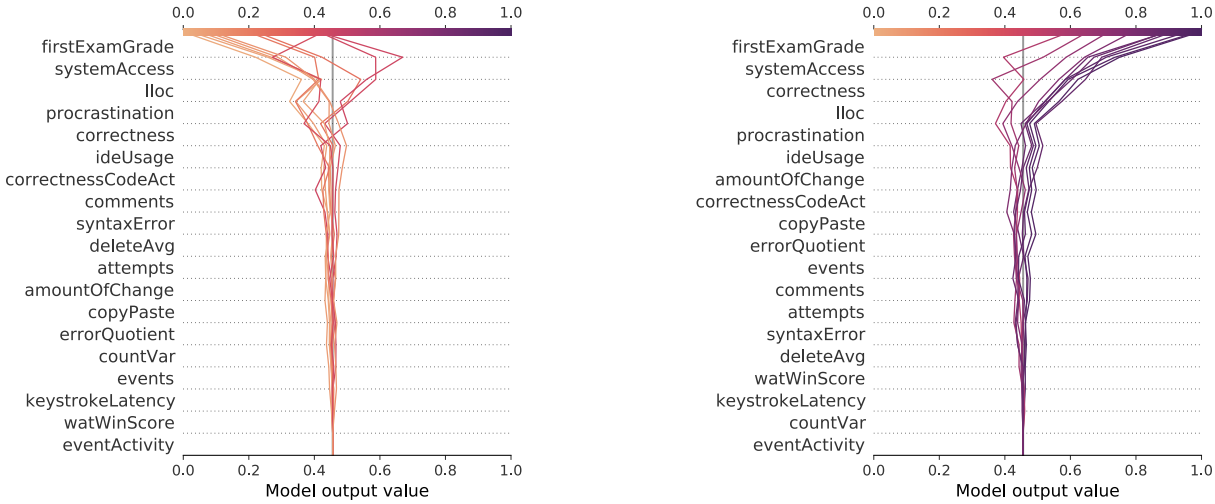
To evaluate possible prediction paths, we cluster the Shapley values of all learners using the well-known k-means algorithm. We use the knee point detection algorithm [62] to automatically find the potential optimal number of clusters. The metrics used to evaluate the maximum curvature point (knee point) [62] were the *mean silhouette score* and *inertia*, as recommended in [24]. After running the k-means algorithm with k ranging from 2 to 10, we found that 5 is the most suitable number of clusters. In other words, we found five different prediction paths, represented by five behavioural patterns that might lead to success and failure, which are shown in Figure 10. These decision plots show the centroids of each cluster. Notice that the centroids in k-means are the averages of the instances inside a cluster. As such, the centroids in this case depict the overall Shapley values (*feature effect*) of the learners in each cluster. In Figure 10, we keep the same feature order in the decision plots, to make it easier to compare the different prediction paths.

Following, we give a brief description of each prediction path that we found (see Figure 10):

- Prediction path 1: students with high chances of passing and who have mostly effective behaviours. They may also have some minor ineffective behaviours.
- Prediction path 2: students with moderate to high chances of passing, who have mostly effective behaviours, but with a different pattern than prediction path 1.
- Prediction path 3: students with a high chance of failing and who have mostly ineffective behaviours. They may also have some slightly effective behaviours.
- Prediction path 4: students whose chances of passing are uncertain. In general, their chances are a bit lower then the base value and, hence, they are borderline cases, potentially unsuccessful. Indeed, these students have moderately effective behaviours; however, they achieved a low grade in the first exam.
- Prediction path 5: students whose chances of passing are uncertain. Their chances are generally a little higher than the base value, and hence, similar to the prediction path 4. They are borderline cases; however, potentially successful ones. Indeed, whilst these learners have some moderate effective and ineffective behaviours, they have a high first exam grade.

Approximately 30.02% of the students follow the prediction path 1, 8.32% follow the prediction path 2, 34.85% follow the prediction path 3, 13.32% follow the prediction path 4, and 13.49% follow the prediction path 5. In other words, 38.34% (30.02% + 8.32%) of the learners have high chances of passing, 34.85% have high chances of failing, and 26.81% (13.32% + 13.49%) are borderline cases, for which the prediction model predicts with moderate to high level of uncertainty.

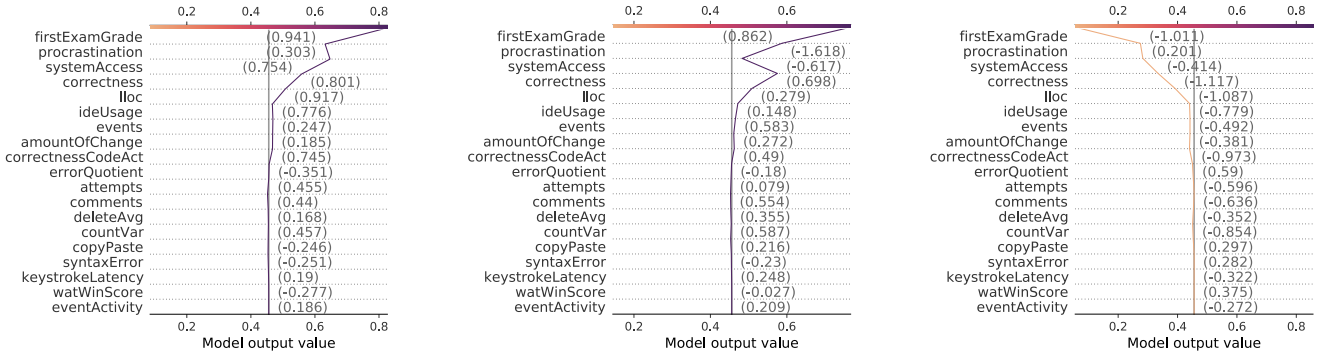
For a better understanding of the prediction paths, we analyse the effective and ineffective behaviours present in each plot from Figure 10. In Figure 10 (a), we can inspect that the learners from this cluster likely made less common errors (e.g., *syntaxError* = -0.25), dealt with the errors better (*errorQuotient* = -0.35) and tended to spend less time to fix errors (*watWinScore* = -0.28), which is a sign that these learners were not struggling to solve the problems. Moreover, they used *copyPaste* (*copyPaste* = -0.25) moderately, which is important for a novice learner. In spite of the importance of knowing that, these behaviours from this cluster have low effect (low Shapley value) in the model's decision. Indeed, the programming behaviours that have highest impact for this prediction path are the fact that the learners from this cluster solved most of the problems from the assignment (*correctness* = 0.80 and *correctnessCodeAct* = 0.76), and spent more than the average time coding in the IDE (*ideUsage* = 0.78). Moreover, the students accessed the system regularly and achieved a moderate to high grade in the first exam (*firstExamGrade* = 0.94). These effective behaviours are the potential



(a) Example of 10 learners with high chances of failing.

(b) Example of 10 learners with high chances of passing.

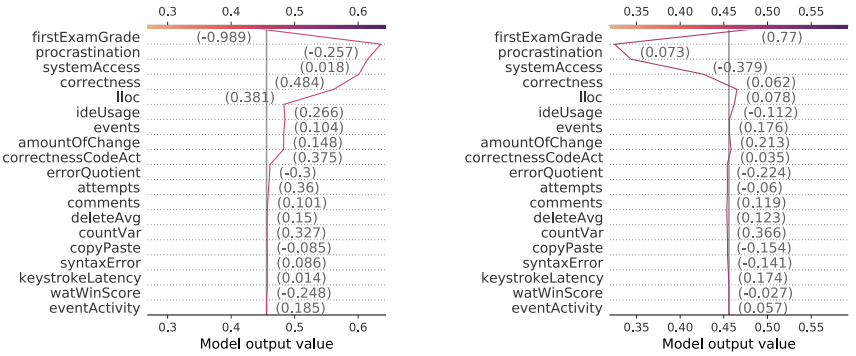
FIGURE 9. **Prediction paths of learners who failed (left) and learners who passed (right).**



(a) Prediction path 1: successful with effective behaviours.

(b) Prediction path 2: successful with mostly effective behaviours, but some ineffective behaviours.

(c) Prediction path 3: unsuccessful and ineffective.



(d) Prediction path 4: potentially unsuccessful borderlines.

(e) Prediction path 5: potentially successful borderlines.

FIGURE 10. **Cluster centroids of prediction paths of our model prediction for a better understanding of effective and ineffective behaviours.**

explanation of why such learners had a probability of passing (close to 80%). The programming behaviours *events* (0.25) and *amountOfChange* (0.19) have also some minor impact in the model's decision. The average value of these features is somewhat expected, as these effective learners do not make many errors, even having a moderate to high number of

attempts (0.46) and correctness (0.80). As a counterexample, a learner who had a moderate to high number of attempts, who solved many problems, and who submitted many code snippets with errors, should have changed her/his code a lot to fix problems, which would have generated many log events. Finally, it is expected that these learners have a low value of *procrastination*, so that there is still room for improvement for the students from this cluster.

In Figure 10 (b) we can observe a similar prediction path (see the trajectory of the coloured line) showed in Figure 10 (a). That is, the learners from this cluster have also high chances of passing, potentially because of similar reasons to the learners who follow prediction path 1. The main difference is that these learners (that follow prediction path 2) have a lower value of *systemAccess* (−0.62) and procrastination (−1.62). However, such a moderately low value of *systemAccess* is likely a positive indicator for this prediction path. Indeed, as the learners solved most of the problems (*correctness* = 0.70), with a low value of *procrastination*, this suggests them solving problems from the assignment as soon as the instructors made them available. Hence, after that, they did not keep accessing the online judge, as they had already finished their assignment.

On the other hand, Figure 10 (c) shows the students who follow the third prediction path. The students from this cluster have more than the average number of code errors (*errorQuotient* = 0.59), and they were not dealing well with the errors (*watWinScore* = 0.38). That is, they potentially were not trying to fix the problems (see the low *amountOfChange* = −0.38), which might explain why they achieved low grades in the first assignment (*correctness* = −1.12) and exam (*firstExamGrade* = −1.01). Additionally, based on the number of attempts (*attempts* = −0.6) and time spent to solve problems (*ideUsage* = −0.78), we can deduce that these learners are neither effective nor resilient in trying to fix code errors. These ineffective behaviours are potential explanations of why the students from this cluster have their average probabilities of passing close to 10%.

Figure 10 (d), the forth prediction path, are learners with a similar trajectory to those following prediction path 1. The main differences are twofold. Firstly, the feature values from this cluster are almost half of those that follow the first prediction path. Overall, they solved half of the questions from the assignments, they have half of the *lloc* value, they spent half of the time that learners from the first cluster spent in solving problems. Secondly, these learners might have some moderate effective behaviours, but achieved a low grade in the first exam (*firstExamGrade* = −0.99). This discrepancy is resulting in the uncertainty of the model for these cases. Indeed, the second reason (low *firstExamGrade*) has a high impact on the model's decision and changed the direction of the prediction to the left, decreasing the learners' chances of passing.

The learners that follow the prediction path 5 (Figure 10 (e)) have average values for almost all programming behaviours, which makes the trajectory of the prediction (looking from the bottom to the top of the plot) close to the grey vertical line (base value). Indeed, the direction starts to change from programming behaviours *ideUsage* and *lloc*, which increase somewhat the chances of passing. This indicates that average values of these 2 features might be effective behaviours for this prediction path. Still, an average *correctness* associated with a moderate to low *systemAccess* and an average *procrastination* is decreasing the learners' chances of passing. A possible reason why an average *correctness* is potentially an ineffective behaviour is that the first assignment has only easy problems and, thus, many learners solved all the questions (for more details, see the *correctness* statistical analysis in Appendix A). Moreover, a moderate/average *procrastination*,[6] without a high correctness, might not seem as an effective behaviour. However, unexpectedly, as an inflection point, a moderate to high *firstExamGrade* changed the direction of this prediction path to the right, raising the overall chances of these learners above the base value. A possible explanation is that these students did not access the IDE regularly (*systemAccess* = −0.38) and may not have solved all the exercises from the programming assignments, not because of lack of knowledge, but because they may already have known programming, i.e., they might have had contact with programming before the CS1 course. Another possibility is simply because of plagiarism in the exam. Notice that this kind of behaviour might confuse the predictive model and bring about false negatives. Such outliers are interesting in themselves to find, to analyse separately (ideally, by an instructor) as they may have quite distinct needs from the rest of the cohort.

Finally, for a deeper analysis of each feature importance based on our model prediction, we make available a link[7] with interactive plots. The shared folder has 10 HTML files with plots for each fold tested in this study. The plots are a combination of individual force plots, rotated 90 degrees and stacked horizontally, and ordered by similarity of SHAP explanation, using the cluster analyses. To illustrate, in Figure 11 we show the first 1000 instances of the first fold (cross-validation). The bold value on the y-axis shows the probability to pass of the student in position 896. Similarly to the explained force plot, the feature values in purple represent a positive effect and the light brown ones a negative effect for this individual student. With such an interactive plot on-hand, the stakeholders (instructors, monitors, coordinators, etc.) can preventively evaluate which behaviour should be stimulated and which should be improved upon, for each student and for groups of learners, since the plot is sorted by similar Shapley values.

H. GLOBAL ANALYSIS

Regarding the importance of the features, Figure 12 (a) presents a bar chart with the average impact (mean of

[6]Notice that a moderate procrastination means that the learner started solving the problems between 4 or 5 days before the deadline (Figure 4).

[7]bit.ly/2PVCCaP

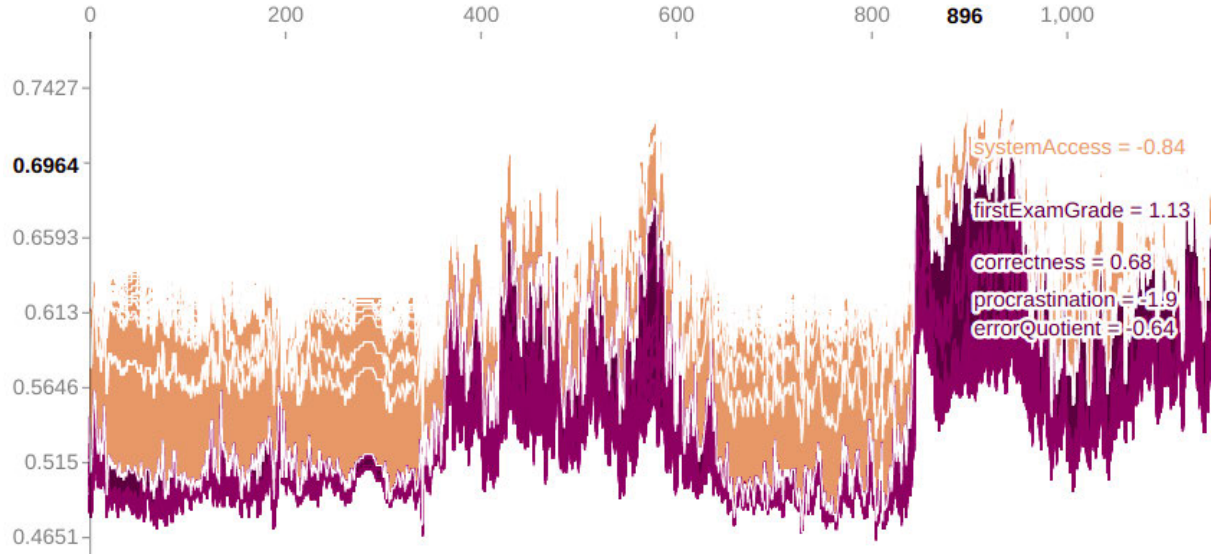


FIGURE 11. Prediction forceplots rotated 90 degrees, stacked horizontally. Here we highlight the 896th instance, showing her/his feature values and contributions.

Shapley values) for each feature, in terms of model output magnitude. As arguably expected, the three most relevant are *firstExamGrade*, *systemAccess*, and *correctness*. This translates into the conclusion that if the learner performs badly in the first exam and in our programming assignment lists, a 'red flag' needs raised. Still, it is important to monitor how regularly students are accessing the online judge, as the number of accesses (*systemAccess*) plays an important role at the beginning of the course. Moreover, the number of logical lines of code (*lloc*) matters in the solution submitted, as *lloc* is the forth most important feature. A potential reason is that the total *lloc* of the solutions sent by the learners for all problems of the first assignment might have an expected value, and the predictive model potentially uncovered the likelihood of the expected value that might be effective or ineffective. Still, we can see in the plot that *procrastination* might be an undesirable behaviour for some students and can influence their performance negatively. Finally, as found by Pereira *et al.* [49], spending more time solving problems (*ideUsage*) and being resilient are positive behaviours. Here, resilience might be measured by the association of *attempts*, *ideUsage*, *lloc*, *errorQuotient*, *syntaxError*, *amountOfChange*, and *watWinScore*. That is, even when the solutions are not correct at first (*errorQuotient*, *syntaxError*), it is important to spend qualitative time (*ideUsage*) trying to fix the error (*attempts*, *amountOfChange*, *watWinScore*) more than once. Notice that such attempts will increase the *lloc* and *countVar*, as these features compute the total number of logical lines of code and variables (respectively) in all submissions, regardless whether accepted or not.

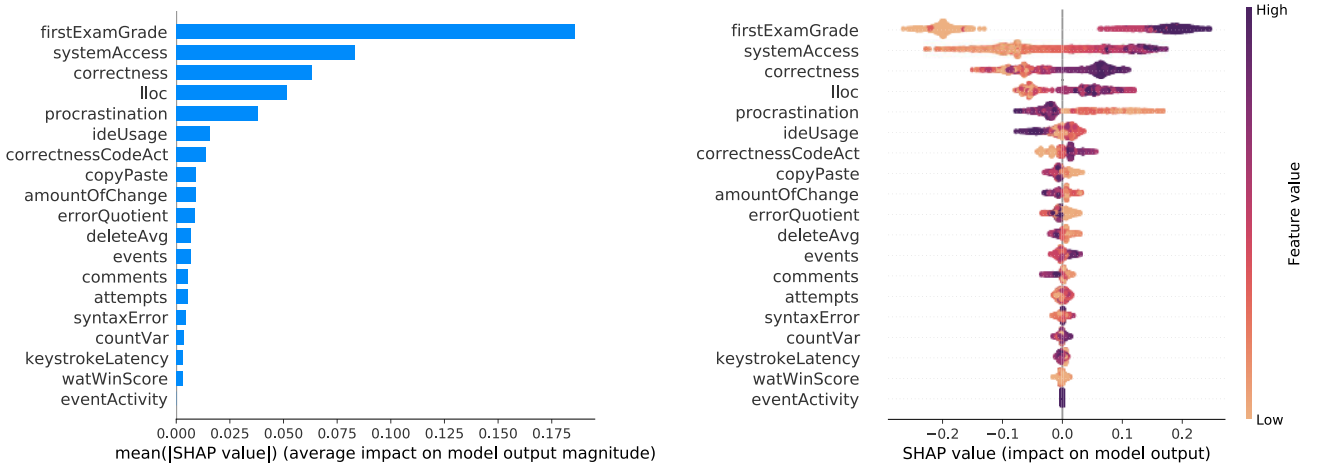
Additionally, it is important to note that the feature effects might be different for different students. To illustrate, whilst general procrastination is associated negatively with performance [66], this effect might be less pronounced or even reversed for some students. In this sense, Figure 12 (b) presents the direction and the distribution of the feature effect. For some features, there are some medium to long tails, meaning that those features might have low global importance, but a high relevance for specific instances. To illustrate, *systemAccess* has a higher total model impact than *procrastination*. Nonetheless, for the instances in which *procrastination* plays an important role (long tail), it has more impact than systemAccess. Thus, *procrastination* impacts a few predictions, by a large amount; whilst *systemAccess* affects almost all predictions, by a smaller amount.

VI. IMPLICATIONS, APPLICATIONS AND IMPACT

Our work enriches the research on programming learning with findings of effective and ineffective early students behaviours (currently considered an open question [11], [50], [53], [58]), and the educational data mining field, with an accurate and explainable ML pipeline that can be useful for early intervention and student self-regulation.

An important finding from our approach is the notion that for different learners, a different set of predictors seem to have an impact on successful learning. As we demonstrated above, even generally undesirable behaviours, like *procrastination* [66] might be more-, or less-harmful, for a particular person. As psychological and educational research typically applies linear modelling [25], such a complex nonlinear interplay has remained undiscovered by prior research applying traditional methods.

Regarding the different features used for prediction in our analysis, we need to emphasise that there are some behaviours that are easier to modify than others. Whilst it is possible to instruct students to avoid procrastination and increase total time investment [6], keystroke latency or the number of deleted characters are less suitable for interventions.



(a) General feature importance.

(b) General feature importance - combining feature importance with feature effects.

FIGURE 12. Summary of features' importance for the model's decision.

For a more generalist analysis for adaptation of instructional decisions, we presented the power of global explanation by the identification and analysis of typical prediction paths. Moreover, our focus not only on global behaviours, but also on individual ones, enabled by visualising and analysing feature effects at single-student granularity level, can be used in an unprecedented variety of pedagogical applications. Indeed, this early prediction, empowered by its explanation, might potentially allow an effective early intervention by stakeholders. To illustrate, our interactive force plots (Figure 11) of each student might be shown to the instructors at the end of the second week of the course, who in turn might create some proactive way of minimising the chances of at-risk students ending up failing. What is more, as the plot is sorted by similar Shapley values, student behaviours might be grouped, for recommendation purposes.

Notice that, in CS1 classes, each student may have a different timing to learn to program. However, in traditional non-personalised classes, all students are treated in the same way. Ideally, students should be challenged to learn as much as they can, taking into consideration their individual learning weaknesses and strengths. For example, a student that solves tasks fast and effortlessly, may be bored and potentially frustrated. One possible solution for that is creating more challenging tasks for the students with high probability of passing. More specifically, more challenging problems may be recommended for students who have low procrastination, solve all the exercises on the assignment, and access the system regularly. Hence, traditional Intelligent Tutoring Systems or Adaptive Educational Hypermedia rule-based approaches [10], [65] can be combined with modern educational data mining and SHAP-based processing for large-scale personalised education.

In addition, for instructors, managers, and educators, a visualisation dashboard, including our force plots, decision plots (and so forth) might contribute for a more formative assessment. As such, not only the learner product is evaluated, but also the process behind, that is, not only their codes are evaluated, but also their learning paths and effort to produce their codes. Formative feedback might be sent to students, to improve student attainment, in response to a request from the literature for such works [15], [31], [38]. For example, an automatic notification might be sent to each learner, showing them their own force plots with their most important behaviours that should be encouraged, and the ones which need improved upon. An individual decision plot might be also sent to students for self-reflection of all analysed behaviours. This would empower the students to better guide their own study. Such metacognitive strategies, which get the students to think about their own learning, have been proven efficient in many areas of education. Indeed, [60] showed that metacognitive strategies may be worth the equivalent of an additional 7x times greater progress than that used in a traditional environment. The study explains that the major reason for such progress is that the learners were aware of their strengths and weaknesses, which motivated them to engage in and improve their learning. Such metacognitive or self-regulatory strategies can also be trained via web-based training [6].

Furthermore, this dashboard can increase the chances of the instructor reflecting and diagnosing potential causes of the students' lack of success. For example, an instructor might explore in the dashboard a plot like in Figure 11, where forceplots are clustered. Using that AI-based information combined with classroom experience, the instructor might schedule a meeting with specific groups, to discuss how certain programming behaviours they are having are potentially jeopardising their learning. In other words, this could amplify the instructor's ability to implement effective interventions.

Indeed, based on such a dashboard we can intervene on many design dimensions [11], such as providing to the learners AI-based information, critique, suggestions, and encouragement. Such intervention content might be shown in our dashboard visually, or through text notification, with the intention of positively affecting their programming behaviours, learning process and outcomes. Moreover, students might also explore visually and interactively their learning process and progress. A dashboard might allow triggered intervention as well, in which a notification or plot might to be shown to the learner in response to her/his actions. Another option would be performing intervention on demand, in which the learner must explicitly require some feedback or suggestion on effective and ineffective behaviours.

Finally, another possibility for interventions is to use the log file data for group formation. As heterogeneity has been proven as beneficial for collaborative learning in many scenarios [40], one pedagogical approach would be to form groups of learners with force plots differing from each other.

VII. LIMITATIONS

It is indisputably important to provide human-friendly feedback to improve the students learning. Here we are going in this direction. However, our model is not 100% accurate and, hence, the model's feedback might not be precise in some cases. That is, it is important to highlight that there are two models being used in this work: the *predictive model* and the *explanatory model*. Both of them have an error component. Thus, the instructor (or other stakeholders) who will receive the feedback from the explanatory model have a crucial role in analysing and interpreting the model's explanation. In this sense, we believe that a first attempt to employ our model should be done by combining humans and our AI, that is, our pipeline could be later validated with a human pipeline of experts. It is noteworthy that the combination of human skills with AI, with the use of hybrid systems, can achieve results superior to those achieved by AI and humans separately [27], [43].

Moreover, in this work, we performed *statistical inference* and not *causal inference*. Our interpretable pipeline using Shapley values, however, offers clear insights to formulate causal hypotheses that could be assessed in future works. In addition, some of the limitations of this work are related to the dataset. In terms of external validation, our sample may not represent the general population. Nonetheless, given that the data was obtained from several years of CS1 courses and students from numerous undergraduate programs, this constraint could be minimised. As a potential internal threat, we did not tackle plagiarism in-depth, and some successful programming behaviours may have been misclassified. We used some features to try to encode plagiarism, such as *attempts*, *eventsActivity*, *copyPaste* and *ideUsage*, since our experience shows that learners who copy codes from other students usually do so as their first attempts (lower number of attempts) and actually spend little time programming (low use of IDE). To confirm that, we plan to carry out further studies.

Finally, the following features had low impact on the model output: *deleteAvg*, *events*, *comments*, *attempts*, *syntaxError*, *countVar*, *keystrokeLatency*, *watWinScore*, and *eventActivity*. This is interesting, since the literature [11], [19], [30], [33], [49], [71] reported that these features are related to student performance. A possible reason is that, again, we are dealing with data from the first two weeks of the course and, hence, some patterns are still not evident yet.

VIII. CONCLUSION AND FUTURE WORKS

In this paper, we developed an explainable ML pipeline that competes in performance with current state-of-the-art (inexplicable) black-box models. We have also shown that there are significant benefits in using fine-grained data-driven code metrics to extract features using insightful algorithms, since this allows, besides predicting student performance early, to analyse behaviours that are related to struggling and successful students. Moreover, we trained our model using data from the first two weeks of classes, allowing early intervention.

For replication purposes, we provide our fine-grained dataset (see Section IV-B). Moreover, for works that want to replicate our work but use only globally relevant features, the most important features for *early prediction* were *firstExamGrade*, *systemAccess*, *correctness*, *lloc*, *procrastination*, *ideUsage*, *correctnessCodeAct*, and *copyPaste* (see Figure 12 (a)). This translates into: if some students perform badly in the first exam and in the early programming assignments, by procrastinating, do not spend appropriate time solving the problems, then a 'red flag' needs raised, as it has likely negative consequences for the students' final performance. Furthermore, we have shown also the local impact of features for each individual student, where less important features could have high relevance for some learners (Figure 12 (b)). As such, researchers that want to replicate this work could consider this local importance of features, additionally to the global one.

Additionally, our high-performance predictive model is explainable, which can facilitate human-AI collaboration towards prescriptive analysis, where the instructors/monitors will have access to individual and collective analysis on which student behaviours should be encouraged, and which ones should be inhibited. On the student side, such analysis can promote self-regulation and awareness of their strengths and their chances for improvement. To illustrate the usefulness of the approach from a student's point of view, they may trust more a recommendation if they understand why they have received it. From the instructors' side, understanding why students are failing or passing would allow them to apply effective efforts to tailor pedagogical materials, instructions and interventions for future classes.

As future work, we will investigate plagiarism behaviour in-depth and its influence in the model's decision. Moreover, we envision to analyse how the data-driven approach used in this paper can model students who begin the course with successful behaviours, but end up with failure behaviours and

grades. Similarly, we will analyse students who change their programming behaviour during the course and the impact of these changes on learning.

Furthermore, a possible extension of the present approach would be to not only create differential explainable models for different learners, but to also investigate whether different situations experienced by the same person have a different impact on the person's learning success, thereby applying a process *perspective on learning* [73]. In order to do so, it would be necessary to collect time series data over a longer period, and to include information about the order of events into the prediction model.

APPENDIX A
DESCRIPTIVE STATISTICS OF PROGRAMMING BEHAVIOURS

Following, we show the explanation and overall analysis results for each feature defined in Table 1 and presented in Figure 4:

- *procrastination*: Here we are analysing the feature before the z-score transformation and multiplication by -1, thus, a higher value means lower procrastination (and vice-versa). In this feature, there is a high positive skewness (skewness $>$ 1.0 and kurtosis $>$ 1.0), indicating asymmetric distribution with a long tail. Indeed, some students solve the problems close to the deadline, however, most of the learners started to solve the problems around 5 days before the deadline (mean = 4.93, median = 4.45). Moreover, we can notice a high variation (std = 3.48, Coefficient of Variation (CV) =.71, and Inter Quartile Range (IQR) = 1.71) in this feature endorsing what we claimed about the heterogeneity of the students' behaviours.
- *amountOfChange*: High positive skewness and kurtosis (skewness $>$ 1.0 and kurtosis $>$ 1.0). This suggests that students tend to change their code slightly between submissions to the same problem (mean =.72, median =.67). This happens typically when students have not had their code accepted in the first submission. A high variation (std =.51, CV =.77, and IQR = 0.55) was observed.
- *eventActivity*: High negative skewness and positive kurtosis (skewness <-1.0 and kurtosis <-1.0). Most students (mean =.69, median =.75) solve the problems with few events (line of logs, see Figure 3). The variation (std =.23, CV =.31, and IQR = 0.18) is moderate to high.
- *attempts*: Symmetric distribution (skewness = 0.31), however with a high kurtosis (kurtosis $>$ 1.0), which can be explained by the presence of outliers: in this case, students who tried many times to solve a given problem. In average, students have attempts of 7.52 and a median of 7.62 per problem, with a moderate to high variation (std = 3.47, CV =.45, IQR = 3.74). The high average and variation in the first two weeks might be explained due to the students learning to manage the online judge system. To deal with the outliers, we applied a root square transformation, to make the distribution normal.
- *comments*: High positive skewness and kurtosis (skewness $>$ 1.0 and kurtosis $>$ 1.0), suggesting that, in general, the students do not document their code (mean = 2.86, median = 3.00), which is expected from novice programmers solving easy problems. Nonetheless, we observe a high variation (std = 2.86, CV =.99, IQR = 4.00) and the presence of outliers. As for attempts, here we also applied the root squared transformation.
- *lloc*: Symmetric distribution (skewness = -0.28, kurtosis = -0.82) with a moderate kurtosis, with a mean similar to the median, indicating a bell-shaped distribution. The average of total *lloc* (mean = 111.71, median = 110.10) is low, as the learners are submitting solutions for problems of arithmetic operations and sequential structures, which require just a few lines of code. Moderate to high variation (mean = 111.71, std = 58.98, CV =.52, IQR = 86.0) is observed.
- *systemAccess*: Most of the students have the number of access to the system in the first 2 weeks of the course concentrated in the lowest values, as the distribution is highly positively skewed (skewness $>$ 1.0 and kurtosis $>$ 1.0), with an average of 32.89 access and high variation (std = 30.98, CV =.94, IQR = 87.00).
- *firstExamGrade*: As the first exams had only 2 problems, students can achieve 0, 5 or 10, if they solve 0, 1 or 2 questions, respectively. That explains the multimodal nature of this distribution, with three potential values of 0, 5 and 10. A different grade is possible when students solve one of the problems partially, receiving a grade proportional to the number of test cases accepted. In average, students solve one problem from the first exam (mean = 5.01, median = 5.00). Notice that the nature of this distribution explains the high variation (std = 4.64, CV =.92, IQR = 10.00).
- *events*: Most students have a lower value of events as the distribution is high positively skewed (skewness $>$ 1.0 and kurtosis $>$ 1.0), with an average of 290.63 events and high variation (std = 180.96, CV =.62, IQR = 201.86).
- *correctness*: Moderate negative skewed distribution (skewness = -0.94 and kurtosis = -0.39), which indicates that most students take the first assignment list seriously and solve the problems. In average, students solved approximately 69% of the problems (mean = 6.91, median = 8.44) in an assignment list comprising 10 or 12 problems. We also observe a moderate variation (std = 3.21, CV =.46, IQR = 4.84).
- *correctnessCodeAct*: Most of students have an average value (mean = 4.68, median = 4.75) of correctnessCodeAct, as the distribution is symmetrical (skewness = -0.14 and kurtosis = -0.95). However, a high variation was observed (std = 2.77, CV = 0.59, IQR = 4.31). Notice that the values of these distributions tend to be

lower than for the correctness distribution, which means that, potentially, some students solved the problems just by copying and pasting, thus, generating only few events.

- *copyPaste*: Highly skewed with a long tail (skewness > 1.0 and kurtosis > 1.0), with an average value of.59 and high variation (std = 0.90, CV = 1.52, IQR = 0.67). As in attempts, here we also applied the root squared transformation due to the presence of outliers (values greater than 1, in this case). Notice that a value greater than 1 means that the learner has pasted more characters than typed (e.g., 50 characters pasted and 10 characters types would lead for a *copyPaste* = 5 (50/10)).
- *syntaxError*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). In average, 29% of the attempts (mean = 0.29, median = 0.24) to solve problems in the first two weeks have this typical error. A high variation was observed (std =.25, CV =.84, IQR =.27).
- *ideUsage*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). In average, students spend 133.93 minutes trying to solve problems in the embedded IDE (mean = 133.93, median = 120.52). A high variation was observed (std = 98.08, CV =.73, IQR = 126.21).
- *keystrokeLatency*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). The keystroke average latency of the learners is 2.59 (mean = 2.59, median = 2.61) and a moderate to high variance was observed (std = 1.04, CV =.73, IQR =.97). As in attempts, here we also applied the root squared transformation, due to the outliers.
- *errorQuotient*: Highly skewed distribution (skewness > 1.0), but with no long tail (kurtosis =.04). We found a low value of *errorQuotient* penalty in pair of errors between submission (mean = 4.19, median = 3.19). A high variation was observed (std = 3.54, CV =.84, IQR = 3.79).
- *watWinScore*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). Students spent a few minutes (mean = 3.34, median = 1.90) between a pair of submissions with errors. A high variation (std = 4.35, CV = 1.30, IQR = 3.99) was observed, due to the presence of some outliers. As in attempts, here we also applied the root squared transformation.
- *countVar*: A moderate to high negative skewed distribution (skewness = −0.72), but with no long tail (kurtosis =.39), with an average of 22.38 (mean = 22.38, median = 3.19) variables in all the code instances submitted by learners. This relatively lower number of variables is due to the easy nature of the initial problem assignments. In addition, a moderate variation was observed (std = 10.56, CV =.47, IQR = 9.89).
- *deleteAvg*: Highly skewed, with a long tail (skewness > 1.0 and kurtosis > 1.0). In average, students make little use of delete (mean = 35.13, median = 29.72). A high variation was observed (std = 26.93, CV =.76, IQR = 27.88), due to the presence of some outliers. As in attempts, here we also applied the root squared transformation. Notice that learners who make more use of delete are potentially rewriting their code more frequently.
- *finalGrade*: Our target variable is a relatively symmetrical (skewness =.2) bimodal distribution (kurtosis > 1.0). Indeed, the left peak of the distribution concentrates most of the failed students and the right peak, the ones that passed. Students achieved an average final grade of 3.93 (mean = 3.93, media = 4.00). A high variation was observed (std = 3.45, CV =.96, IQR = 6.74).

ACKNOWLEDGMENT

This research, carried out within the scope of the Samsung-UFAM Project for Education and Research (SUPER), according to Article 48 of Decree no. 6.008/2006 (SUFRAMA), was partially funded by Samsung Electronics of Amazonia Ltda., under the terms of Federal Law no. 8.387/1991, through agreements 001/2020 and 003/2019, signed with Federal University of Amazonas and FAEPI, Brazil. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and the Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq) through the support to Elaine Oliveira (Process 308513/2020-7).

REFERENCES

[1] A. Ahadi, R. Lister, H. Haapala, and A. Vihavainen, "Exploring machine learning methods to automatically identify students in need of assistance," in *Proc. 11th Annu. Int. Conf. Int. Comput. Educ. Res.*, Aug. 2015, pp. 121–130.

[2] A. Ahadi, R. Lister, and A. Vihavainen, "On the number of attempts students made on some online programming exercises during semester and their subsequent performance on final exam questions," in *Proc. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, Jul. 2016, pp. 218–223.

[3] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, no. 7, Jul. 2015, Art. no. e0130140.

[4] R. S. Baker and P. S. Inventado, "Educational data mining and learning analytics," in *Learning Analytics*. New York, NY, USA: Springer, 2014, pp. 61–75.

[5] G. E. Batista, R. C. Prati, and M. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explor. Newslett.*, vol. 6, no. 1, pp. 20–29, 2004.

[6] H. Bellhäuser, T. Lösch, C. Winter, and B. Schmitz, "Applying a web-based training to foster self-regulated learning—Effects of an intervention for large numbers of participants," *Internet Higher Educ.*, vol. 31, pp. 87–100, Oct. 2016.

[7] B. Berendt, A. Littlejohn, and M. Blakemore, "AI in education: Learner choice and fundamental rights," *Learn., Media Technol.*, vol. 45, no. 3, pp. 312–324, Jul. 2020.

[8] J. L. Bez, N. A. Tonin, and P. R. Rodegheri, "URI online judge academic: A tool for algorithms and programming classes," in *Proc. 9th Int. Conf. Comput. Sci. Educ.*, Aug. 2014, pp. 149–152.

[9] D. Boulanger and V. Kumar, "SHAPed automated essay scoring: Explaining writing features' contributions to English writing organization," in *Proc. Int. Conf. Intell. Tutoring Syst.* Cham, Switzerland: Springer, 2020, pp. 68–78.

[10] E. Brown, A. I. Cristea, C. Stewart, and T. Brailsford, "Patterns in authoring of adaptive educational hypermedia: A taxonomy of learning styles," *J. Educ. Technol. Soc.*, vol. 8, no. 3, pp. 77–90, 2005.

[11] A. Carter, C. Hundhausen, and D. Olivares, "Leveraging the integrated development environment for learning analytics," in *The Cambridge Handbook of Computing Education Research*. Cambridge, U.K.: Cambridge Univ. Press, 2019, ch 23, pp. 679–706.

[12] K. Castro-Wunsch, A. Ahadi, and A. Petersen, "Evaluating neural networks as a method for identifying students in need of assistance," in *Proc. ACM SIGCSE Tech. Symp. Comput. Sci. Educ.*, Mar. 2017, pp. 111–116.

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.

[14] F. Chollet, *Deep Learning With Python*. Simon and Schuster, 2017.

[15] A. N. Crişan, "Case study on the importance of formative assessment in stimulating student motivation for learning and increasing the efficiency of the educational process," *J. Educ. Sci. Psychol.*, vol. 7, no. 1, pp. 20–25, 2017.

[16] F. Dwan, E. H. T. Oliveira, and D. Fernandes, "Predição de zona de aprendizagem de alunos de introdução à programação em ambientes de correção automática de código," in *Proc. Brazilian Symp. Comput. Educ. (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 28, 2017, p. 1507.

[17] L. Echeverría, R. Cobos, L. Machuca, and I. Claros, "Using collaborative learning scenarios to teach programming to non-CS majors," *Comput. Appl. Eng. Educ.*, vol. 25, no. 5, pp. 719–731, 2017.

[18] J. Edwards, J. Leinonen, and A. Hellas, "A study of keystroke data in two contexts: Written language and programming language influence predictability of learning outcomes," in *Proc. 51st ACM Tech. Symp. Comput. Sci. Educ.*, 2020, pp. 413–419.

[19] S. H. Edwards, J. Snyder, M. A. Pérez-Quiñones, A. Allevato, D. Kim, and B. Tretola, "Comparing effective and ineffective behaviors of student programmers," in *Proc. 5th Int. Workshop Comput. Educ. Res. Workshop*, 2009, pp. 3–14.

[20] A. Estey and Y. Coady, "Can interaction patterns with supplemental study tools predict outcomes in CS1?" in *Proc. 2016 ACM Conf. Innov. Technol. Comput. Sci. Educ.*, 2016, pp. 236–241.

[21] S. C. Fonseca, E. H. T. Oliveira, F. D. Pereira, D. Fernandes, and L. S. G. Carvalho, "Adaptação de um método preditivo para inferir o desempenho de alunos de programação," in *Proc. Brazilian Symp. Comput. Educ. (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 30, 2019, p. 1651.

[22] S. C. Fonseca, F. D. Pereira, E. H. T. Oliveira, D. B. F. Oliveira, L. S. G. Carvalho, and A. I. Cristea, "Automatic subject-based contextualisation of programming assignment lists," presented at the 13th Int. Conf. Educ. Data Mining (EDM), Jul. 2020.

[23] H. L. Fwa, "Predicting non-completion of programming exercises using action logs and keystrokes," in *Proc. Int. Symp. Educ. Technol. (ISET)*, Jul. 2019, pp. 271–275.

[24] A. Géron, *Hands-On Machine Learning With Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[25] S. J. Guastello, M. Koopmans, and D. Pincus, *Chaos and Complexity in Psychology: The Theory of Nonlinear Dynamical Systems*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[26] O. Hazzan, T. Lapidot, and N. Ragonis, *Guide to Teaching Computer Science*. Cham, Switzerland: Springer, 2014.

[27] K. Holstein, V. Aleven, and N. Rummel, "A conceptual framework for human–AI hybrid adaptivity in education," in *Proc. Int. Conf. Artif. Intell. Educ.* Cham, Switzerland: Springer, 2020, pp. 240–254.

[28] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, and D. Toll, "Educational data mining and learning analytics in programming: Literature review and case studies," in *Proc. ITiCSE Work. Group Rep.*, 2015, pp. 41–63.

[29] C. M. Intisar and Y. Watanobe, "Classification of online judge programmers based on rule extraction from self organizing feature map," in *Proc. 9th Int. Conf. Awareness Sci. Technol. (iCAST)*, Sep. 2018, pp. 313–318.

[30] M. C. Jadud, "Methods and tools for exploring novice compilation behaviour," in *Proc. 2nd Int. Workshop Comput. Educ. Res.*, 2006, pp. 73–84.

[31] M. W. Keefer, S. E. Wilson, H. Dankowicz, and M. C. Loui, "The importance of formative assessment in science and engineering ethics education: Some evidence and practical advice," *Sci. Eng. Ethics*, vol. 20, no. 1, pp. 249–260, Mar. 2014.

[32] J. Lagus, K. Longi, A. Klami, and A. Hellas, "Transfer-learning methods in programming course outcome prediction," *ACM Trans. Comput. Educ.*, vol. 18, no. 4, pp. 1–18, Nov. 2018.

[33] J. Leinonen, K. Longi, A. Klami, and A. Vihavainen, "Automatic inference of programming performance and experience from typing patterns," in *Proc. 47th ACM Tech. Symp. Comput. Sci. Educ.*, 2016, pp. 132–137.

[34] S. Lipovetsky and M. Conklin, "Analysis of regression in game theory approach," *Appl. Stochastic Models Bus. Ind.*, vol. 17, no. 4, pp. 319–330, 2001.

[35] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, "From local explanations to global understanding with explainable AI for trees," *Nature Mach. Intell.*, vol. 2, no. 1, pp. 56–67, 2020.

[36] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4765–4774.

[37] S. M. Lundberg, B. Nair, M. S. Vavilala, M. Horibe, M. J. Eisses, T. Adams, D. E. Liston, D. K.-W. Low, S.-F. Newman, J. Kim, and S.-I. Lee, "Explainable machine-learning predictions for the prevention of hypoxaemia during surgery," *Nature Biomed. Eng.*, vol. 2, no. 10, pp. 749–760, Oct. 2018.

[38] I. Y. C. Menéndez, M. A. C. Napa, M. L. M. Moreira, and G. G. V. Zambrano, "The importance of formative assessment in the learning teaching process," *Int. J. Social Sci. Humanities*, vol. 3, no. 2, pp. 238–249, Aug. 2019.

[39] C. Molnar, *Interpretable Machine Learning*. Abu Dhabi, United Arab Emirates: Lulu, 2019.

[40] A. Müller, H. Bellhäuser, J. Konert, and R. Röpke, "Effects of group formation on student satisfaction and performance: A field experiment," *Small Group Res.*, vol. 1, Feb. 2021, Art. no. 1046496420988592.

[41] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, "Definitions, methods, and applications in interpretable machine learning," *Proc. Nat. Acad. Sci. USA*, vol. 116, no. 44, pp. 22071–22080, Oct. 2019.

[42] J. Otero, L. Junco, R. Suárez, A. Palacios, I. Couso, and L. Sánchez, "Finding informative code metrics under uncertainty for predicting the pass rate of online courses," *Inf. Sci.*, vol. 373, pp. 42–56, Dec. 2016.

[43] R. Paiva and I. I. Bittencourt, "Helping teachers help their students: A human-AI hybrid approach," in *Proc. Int. Conf. Artif. Intell. Educ.* Cham, Switzerland: Springer, 2020, pp. 448–459.

[44] F. Pedregosa, V. Gaël, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[45] F. D. Pereira, S. C. Fonseca, E. H. T. Oliveira, D. B. F. Oliveira, A. I. Cristea, and L. S. G. Carvalho, "Deep learning for early performance prediction of introductory programming students: A comparative and explanatory study," *Brazilian J. Comput. Educ.*, vol. 28, no. 1, pp. 723–749, 2020.

[46] F. D. Pereira, E. H. T. Oliveira, A. I. Cristea, D. Fernandes, L. Silva, G. Aguiar, and M. Alshehri, "Early dropout prediction for programming courses supported by online judges," in *Proc. Int. Conf. Artif. Intell. Educ.* Cham, Switzerland: Springer, 2019, pp. 67–72.

[47] F. D. Pereira, E. H. T. Oliveira, D. Fernandes, and A. Cristea, "Early performance prediction for CS1 course students using a combination of machine learning and an evolutionary algorithm," in *Proc. IEEE 19th Int. Conf. Adv. Learn. Technol. (ICALT)*, vol. 2161, Jul. 2019, pp. 183–184.

[48] F. D. Pereira, E. H. T. Oliveira, D. Fernandes, H. Junior, and L. S. G. Carvalho, "Otimização e automação da predição precoce do desempenho de alunos que utilizam juízes online: Uma abordagem com algoritmo genético," in *Brazilian Symp. Comput. Educ. (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 30, 2019, p. 1451.

[49] F. D. Pereira, E. H. T. Oliveira, D. B. F. Oliveira, A. I. Cristea, L. S. G. Carvalho, S. C. Fonseca, A. Toda, and S. Isotani, "Using learning analytics in the amazonas: Understanding Students' behaviour in introductory programming," *Brit. J. Educ. Technol.*, vol. 51, no. 4, pp. 955–972, Jul. 2020.

[50] F. D. Pereira, L. M. Souza, E. H. T. Oliveira, D. B. F. Oliveira, and L. S. G. Carvalho, "Predição de desempenho em ambientes computacionais para turmas de programação: Um mapeamento sistemático da literatura," in *Proc. Anais do 31th Simpósio Brasileiro de Informática na Educação*, 2020, pp. 1673–1682.

[51] J. Petit, O. Giménez, and S. Roura, "Jutge.org: An educational programming judge," in *Proc. 43rd ACM Tech. Symp. Comput. Sci. Educ.*, 2012, pp. 445–450.

[52] K. Quille and S. Bergin, "Programming: Predicting Student success early in CS1. A re-validation and replication study," in *Proc. 23rd Annu. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, Jul. 2018, pp. 15–20.
[53] K. Quille and S. Bergin, "CS1: How will they do? How can we help? A decade of research and practice," *Comput. Sci. Educ.*, vol. 29, nos. 2–3, pp. 254–282, Jul. 2019.
[54] I. Rahwan *et al.*, "Machine behaviour," *Nature*, vol. 568, no. 7753, pp. 477–486, 2019.
[55] A. Rai, "Explainable AI: From black box to glass box," *J. Acad. Marketing Sci.*, vol. 48, no. 1, pp. 137–141, Jan. 2020.
[56] M. A. Revilla, S. Manzoor, and R. Liu, "Competitive learning in informatics: The UVA online judge experience," *Olympiads Informat.*, vol. 2, no. 10, pp. 131–148, 2008.
[57] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?' explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144.
[58] A. V. Robins, "Novice programmers and introductory programming," in *The Cambridge Handbook of Computing Education Research*. Cambridge, U.K.: Cambridge Univ. Press, 2019, ch 12, pp. 327–376.
[59] C. Romero and S. Ventura, "Guest editorial: Special issue on early prediction and supporting of learning performance," *IEEE Trans. Learn. Technol.*, vol. 12, no. 2, pp. 145–147, Apr. 2019.
[60] C. A. Rowland, "The effect of testing versus restudy on retention: A meta-analytic review of the testing effect," *Psychol. Bull.*, vol. 140, no. 6, p. 1432, 2014.
[61] B. L. Santana and R. A. Bittencourt, "Increasing motivation of CS1 non-majors through an approach contextualized by games and media," in *Proc. IEEE Frontiers Educ. Conf. (FIE)*, Oct. 2018, pp. 1–9.
[62] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan, "Finding a 'kneedle' in a haystack: Detecting knee points in system behavior," in *Proc. 31st Int. Conf. Distrib. Comput. Syst. Workshops*, Jun. 2011, pp. 166–171.
[63] L. S. Shapley, *A Value for n-Person Games*. Princeton, NJ, USA: Princeton Univ. Press, 1953, pp. 307–317.
[64] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," 2017, *arXiv:1704.02685*. [Online]. Available: http://arxiv.org/abs/1704.02685
[65] N. V. Stash, A. I. Cristea, and P. M. De Bra, "Authoring of learning styles in adaptive hypermedia: Problems and solutions," in *Proc. 13th Int. World Wide Web Conf. Alternate Track Papers Posters*, 2004, pp. 114–123.
[66] S. Piers, "The nature of procrastination: A meta-analytic and theoretical review of quintessential self-regulatory failure," *Psychol. Bull.*, vol. 133, no. 1, pp. 65–94, 2007.
[67] Q. Sun, J. Wu, and K. Liu, "Toward understanding Students' learning performance in an object-oriented programming course: The perspective of program quality," *IEEE Access*, vol. 8, pp. 37505–37517, 2020.
[68] N. Tomasevic, N. Gvozdenovic, and S. Vranes, "An overview and comparison of supervised data mining techniques for Student exam performance prediction," *Comput. Educ.*, vol. 143, Jan. 2020, Art. no. 103676.
[69] I. Tomek, "Two modifications of CNN," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. SMCS-6, no. 11, p. 769—772, Nov. 1976.
[70] C. Watson and F. W. B. Li, "Failure rates in introductory programming revisited," in *Proc. Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, 2014, pp. 39–44.
[71] C. Watson, F. W. Li, and J. L. Godwin, "Predicting performance in an introductory programming course by logging and analyzing student programming behavior," in *Proc. IEEE 13th Int. Conf. Adv. Learn. Technol.*, Jul. 2013, pp. 319–323.
[72] D. H. Wolpert, "The supervised learning no-free-lunch theorems," in *Soft Computing and Industry*. London, U.K.: Springer, 2002, pp. 25–42.
[73] B. J. Zimmerman, "Investigating self-regulation and motivation: Historical background, methodological developments, and future prospects," *Amer. Educ. Res. J.*, vol. 45, no. 1, pp. 166–183, 2008.

FILIPE DWAN PEREIRA received the B.S. degree in computer science from the Federal University of Roraima and the M.S. degree in computer science from the Federal University of Amazonas, where he is currently pursuing the Ph.D. degree in artificial intelligence applied to education. Since 2013, he has been an Auxiliary Professor with the Department of Computer Science, Federal University of Roraima. His research interests include education data mining, learning analytics, artificial intelligence, machine learning, big data, computing in education, and information systems.

SAMUEL C. FONSECA is currently an Undergraduate Student in computer engineering with the Federal University of Amazonas. He also works as a Software Developer with SIDIA, the largest research and development institute in Latin America. His research interests include machine learning, computer vision, and natural language processing.

ELAINE H. T. OLIVEIRA received the Diploma degree in computer science from the University of São Paulo, Brazil, and the Ph.D. degree at the Graduate Program in informatics and education from the Federal University of Rio Grande do Sul, Brazil, in 2011. Since 2002, she has been a Professor with the Institute of Computing, Federal University of Amazonas. She was an Associate Editor of the *Brazilian Journal of Computers in Education*, from 2019 to 2021. Her present research interest includes studying students' behavior as they learn how to program, using learning paths, and data-driven approaches. The data is collected by the interaction of the students with a self-devised Online Judge and Learning Management Systems. The goals of her research are to predict outcomes, help decision making, and provide adaptive learning through learning analytics.

ALEXANDRA I. CRISTEA (Senior Member, IEEE) is currently a Professor and the Head of the Artificial Intelligence and Human Systems (AIHS) Group and the Deputy Head with the Computer Science Department, Durham University. She is also an Honorary Professor with the Computer Science Department, University of Warwick. Her research interests include web science, learning analytics, user modeling and personalization, semantic web, social web, authoring, with over 300 articles on these subjects (over 4000 citations on Google Scholar, H-index 35). She was classified within the top 50 researchers in the world in the area of educational computer-based research according to Microsoft Research. She has been highly active and has an influential role in international research projects. She has led various projects. She has been a keynote/invited speaker, an organizer, a co-organizer, a panelist, and a program committee member of various conferences in her research field. She is a member of the Editorial Board of the IEEE Transactions on Learning Technologies, an Executive Peer Reviewer of the IEEE LTTF Education Technology and Society Journal, and an Associate Editor of *Frontiers in Artificial Intelligence*.

HENRIK BELLHÄUSER developed a web-based training to promote the self-regulatory competence of students in his doctoral thesis. He is currently a Postdoctoral Researcher in educational psychology with the Johannes Gutenberg-University Mainz, Germany. His current research interests include optimised group formation and collaborative digital learning.

LUIZ RODRIGUES received the B.Sc. degree in computer science from Philadelphia University Center—UniFil, in 2016, and the M.Sc. degree in computer science from Londrina State University (UEL), in 2018. He is currently pursuing the Ph.D. degree in computer science and computational mathematics with the University of São Paulo (USP). He is also a Researcher with the Laboratory of Applied Computing to Education and Advanced Social Technology (CAEd), USP. He taught at multiples minicourses as a volunteer, worked as an invited professor both on undergraduate and graduate levels, and revised papers for both national and international events. His main research interests include gamification, user modeling, educational data mining, educational games, and procedural content generation.

DAVID B. F. OLIVEIRA received the Ph.D. degree in computer science from the Federal University of Minas Gerais, in 2010. He is currently a Professor with the Institute of Computing, Federal University of Amazonas, Brazil, where he works as a Researcher, a Professor, and an Advisor in undergraduate. He has experience in information retrieval and informatics in education areas, having worked on the following research topics, such as search for structured content, online judges, gamification of educational systems, and web development. He is also the Development Team Manager of the CodeBench System, which is an online judge which automatically grades the programming assignments submitted by students.

SEIJI ISOTANI (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from the University of Sao Paulo, Brazil, and the Ph.D. degree in information engineering from Osaka University, Japan.

He is currently a Full Professor in computer science and learning technology with the Institute of Mathematics and Computer Science, University of Sao Paulo. His research career has been devoted to the conception, design, development, testing and deployment of intelligent, and collaborative educational systems using ontologies and other semantic technologies. His scientific and social missions converge into a single objective to enable the realization of Anytime, Anywhere, Anybody Learning (AAAL) by developing cutting-edge technology. He is the Co-Founder of two startups (MeuTutor and Linkn), which have won several innovation awards in the fields of education and the semantic web. He has published over a 100 scientific articles, books, and book chapters on educational technology and the semantic web. His mains research interests include gamification, ontological engineering, computer-supported collaborative learning (CSCL), artificial intelligence in education (AIED), and technology-enhanced learning.

LEANDRO S. G. CARVALHO received the B.S. degree in electronic engineering from the Technological Institute of Aeronautics, Brazil, in 2000, and the M.S. and D.S. degrees in informatics from the Federal University of Amazonas, Brazil, in 2004 and 2011, respectively. Since 2006, he has been a Professor with the Institute of Computing, Federal University of Amazonas. His research interests include computer science education and educational data mining. He has been an organizer, a co-organizer, and a program committee member of various conferences in his research field.